IsarMathLib

Sławomir Kołodyński, Daniel de la Concepción Sáez

May 17, 2025

Abstract

This is the proof document of the IsarMathLib project version 1.33.0. IsarMathLib is a library of formalized mathematics for Isabelle2025 (ZF logic).

Contents

1 Introduction to the IsarMathLib project

theory `Introduction` **imports** `ZF.equalities`

begin

This theory does not contain any formalized mathematics used in other theories, but is an introduction to IsarMathLib project.

1.1 How to read IsarMathLib proofs - a tutorial

Isar (the Isabelle's formal proof language) was designed to be similar to the standard language of mathematics. Any person able to read proofs in a typical mathematical paper should be able to read and understand Isar proofs without having to learn a special proof language. However, Isar is a formal proof language and as such it does contain a couple of constructs whose meaning is hard to guess. In this tutorial we will define a notion and prove an example theorem about that notion, explaining Isar syntax along the way. This tutorial may also serve as a style guide for IsarMathLib contributors. Note that this tutorial aims to help in reading the presentation of the Isar language that is used in IsarMathLib proof document and HTML rendering on the FormalMath.org site, but does not teach how to write proofs that can be verified by Isabelle. This presentation is different than the source processed by Isabelle (the concept that the source and presentation look different should be familiar to any LaTeX user). To learn how to write Isar proofs one needs to study the source of this tutorial as well.

The first thing that mathematicians typically do is to define notions. In Isar this is done with the `definition` keyword. In our case we define a notion of two sets being disjoint. We will use the infix notation, i.e. the string `{is disjoint with}` put between two sets to denote our notion of disjointness. The left side of the ≡ symbol is the notion being defined, the right side says how we define it. In Isabelle/ZF `0` is used to denote both zero (of natural numbers) and the empty set, which is not surprising as those two things are the same in set theory.

definition
  `AreDisjoint (`**infix** `{is disjoint with} 90)` **where**
  `A {is disjoint with} B ≡ A ∩ B = 0`

We are ready to prove a theorem. Here we show that the relation of being disjoint is symmetric. We start with one of the keywords "theorem", "lemma" or "corollary". In Isar they are synonymous. Then we provide a name for the theorem. In standard mathematics theorems are numbered. In Isar we can do that too, but it is considered better to give theorems meaningful names. After the "shows" keyword we give the statement to show. The ⟷ symbol denotes the equivalence in Isabelle/ZF. Here we want to show that "A is disjoint with B iff and only if B is disjoint with A". To prove this fact we show two implications - the first one that `A {is disjoint with} B` implies `B {is disjoint with} A` and then the converse one. Each of these implications is formulated as a statement to be proved and then proved in a subproof like a mini-theorem. Each subproof uses a proof block to show the implication. Proof blocks are delimited with curly brackets in Isar. Proof block is one of the constructs that does not exist in informal mathematics, so it may be confusing. When reading a proof containing a proof block I suggest to focus first on what is that we are proving in it. This can be done by looking at the first line or two of the block and then at the last statement. In our case the block starts with "assume `A {is disjoint with} B` and the last statement is "then have `B {is disjoint with} A`". It is a typical pattern when someone needs to prove an implication: one assumes the antecedent and then shows that the consequent follows from this assumption. Implications are denoted with the ⟶ symbol in Isabelle. After we prove both implications we collect them using the "moreover" construct. The keyword "ultimately" indicates that what follows is the conclusion of the statements collected with "moreover". The "show" keyword is like "have", except that it indicates that we have arrived at the claim of the theorem (or a subproof).

```
theorem disjointness_symmetric:
  shows A {is disjoint with} B ⟷ B {is disjoint with} A
proof -
  have A {is disjoint with} B ⟶ B {is disjoint with} A
  proof -
    { assume A {is disjoint with} B
      then have A ∩ B = 0 using AreDisjoint_def by simp
      hence B ∩ A = 0 by auto
      then have  B {is disjoint with} A
        using AreDisjoint_def by simp
    } thus thesis by simp
  qed
  moreover have B {is disjoint with} A ⟶ A {is disjoint with} B
  proof -
    { assume B {is disjoint with} A
      then have B ∩ A = 0 using AreDisjoint_def by simp
      hence A ∩ B = 0 by auto
      then have  A {is disjoint with} B
        using AreDisjoint_def by simp
    } thus thesis by simp
  qed
  ultimately show thesis by blast
qed
```

1.2 Overview of the project

The `Fol1`, `ZF1` and `Nat_ZF_IML` theory files contain some background material that is needed for the remaining theories.

`Order_ZF` and `Order_ZF_1a` reformulate material from standard Isabelle's `Order` theory in terms of non-strict (less-or-equal) order relations. `Order_ZF_1` on the other hand directly continues the `Order` theory file using strict order relations (less and not equal). This is useful for translating theorems from Metamath.

In `NatOrder_ZF` we prove that the usual order on natural numbers is linear.

The `func1` theory provides basic facts about functions. `func_ZF` continues this development with more advanced topics that relate to algebraic properties of binary operations, like lifting a binary operation to a function space, associative, commutative and distributive operations and properties of functions related to order relations. `func_ZF_1` is about properties of functions related to order relations.

The standard Isabelle's `Finite` theory defines the finite powerset of a set as a certain "datatype" (?) with some recursive properties. IsarMathLib's `Finite1` and `Finite_ZF_1` theories develop more facts about this notion. These two theories are obsolete now. They will be gradually replaced by an approach based on set theory rather than tools specific to Isabelle. This approach is presented in `Finite_ZF` theory file.

In `FinOrd_ZF` we talk about ordered finite sets.

The `EquivClass1` theory file is a reformulation of the material in the standard Isabelle's `EquivClass` theory in the spirit of ZF set theory.

`FiniteSeq_ZF` discusses the notion of finite sequences (a.k.a. lists).

`InductiveSeq_ZF` provides the definition and properties of (what is known in basic calculus as) sequences defined by induction, i. e. by a formula of the form $a_0 = x,\ a_{n+1} = f(a_n)$.

`Fold_ZF` shows how the familiar from functional programming notion of fold can be interpreted in set theory.

`Partitions_ZF` is about splitting a set into non-overlapping subsets. This is a common trick in proofs.

`Semigroup_ZF` treats the expressions of the form $a_0 \cdot a_1 \cdot .. \cdot a_n$, (i.e. products of finite sequences), where "$\cdot$" is an associative binary operation.

`CommutativeSemigroup_ZF` is another take on a similar subject. This time we consider the case when the operation is commutative and the result of depends only on the set of elements we are summing (additively speaking), but not the order.

The `Topology_ZF` series covers basics of general topology: interior, closure, boundary, compact sets, separation axioms and continuous functions.

`Group_ZF`, `Group_ZF_1`, `Group_ZF_1b` and `Group_ZF_2` provide basic facts of the group theory. `Group_ZF_3` considers the notion of almost homomorphisms that is nedeed for the real numbers construction in `Real_ZF`.

The `TopologicalGroup` connects the `Topology_ZF` and `Group_ZF` series and starts the subject of topological groups with some basic definitions and facts.

In `DirectProduct_ZF` we define direct product of groups and show some its basic properties.

The `OrderedGroup_ZF` theory treats ordered groups. This is a suprisingly large theory for such relatively obscure topic.

`Ring_ZF` defines rings. `Ring_ZF_1` covers the properties of rings that are specific to the real numbers construction in `Real_ZF`.

The `OrderedRing_ZF` theory looks at the consequences of adding a linear order to the ring algebraic structure.

`Field_ZF` and `OrderedField_ZF` contain basic facts about (you guessed it) fields and ordered fields.

`Int_ZF_IML` theory considers the integers as a monoid (multiplication) and an abelian ordered group (addition). In `Int_ZF_1` we show that integers form a commutative ring. `Int_ZF_2` contains some facts about slopes (almost homomorphisms on integers) needed for real numbers construction, used in `Real_ZF_1`.

In the `IntDiv_ZF_IML` theory we translate some properties of the integer quotient and reminder functions studied in the standard Isabelle's `IntDiv_ZF` theory to the notation used in IsarMathLib.

The `Real_ZF` and `Real_ZF_1` theories contain the construction of real numbers based on the paper [2] by R. D. Arthan (not Cauchy sequences, not Dedekind sections). The heavy lifting is done mostly in `Group_ZF_3`, `Ring_ZF_1` and `Int_ZF_2`. `Real_ZF` contains the part of the construction that can be done starting from generic abelian groups (rather than additive group of integers). This allows to show that real numbers form a ring. `Real_ZF_1` continues the construction using properties specific to the integers and showing that real numbers constructed this way form a complete ordered field.

`Cardinal_ZF` provides a couple of theorems about cardinals that are mostly used for studying properties of topological properties (yes, this is kind of meta). The main result (proven without AC) is that if two sets can be injectively mapped into an infinite cardinal, then so can be their union. There is also a definition of the Axiom of Choice specific for a given cardinal (so that the choice function exists for families of sets of given cardinality). Some properties are proven for such predicates, like that for finite families of sets the choice function always exists (in ZF) and that the axiom of choice for a larger cardinal implies one for a smaller cardinal.

`Group_ZF_4` considers conjugate of subgroup and defines simple groups. A nice theorem here is that endomorphisms of an abelian group form a ring. The first isomorphism theorem (a group homomorphism h induces an isomorphism between the group divided by the kernel of h and the image of h) is proven.

Turns out given a property of a topological space one can define a local version of a property in general. This is studied in the the `Topology_ZF_properties_2` theory and applied to local versions of the property of being finite or compact or Hausdorff (i.e. locally finite, locally compact, locally Hausdorff). There are a couple of nice applications, like one-point compactification that allows to show that every locally compact Hausdorff space is regular. Also there are some results on the interplay between hereditability of a property and local properties.

For a given surjection $f : X \to Y$, where X is a topological space one can consider the weakest topology on Y which makes f continuous, let's call it a quotient topology generated by f. The quotient topology generated by an equivalence relation r on X is actually a special case of this setup, where f is the natural projection of X on the quotient X/r. The properties of these two ways of getting new topologies are studied in `Topology_ZF_8` theory. The main result is that any quotient topology generated by a function is homeomorphic to a topology given by an equivalence relation, so these two approaches to quotient topologies are kind of equivalent.

As we all know, automorphisms of a topological space form a group. This fact is proven in `Topology_ZF_9` and the automorphism groups for co-cardinal, included-set, and excluded-set topologies are identified. For order topologies it is shown that order isomorphisms are homeomorphisms of the topology induced by the order. Properties preserved by continuous functions are studied and as an application it is shown for example that quotient topological spaces of compact (or connected) spaces are compact (or connected, resp.)

The `Topology_ZF_10` theory is about products of two topological spaces. It is proven that if two spaces are T_0 (or T_1, T_2, regular, connected) then their product is as well.

Given a total order on a set one can define a natural topology on it generated by taking the rays and intervals as the base. The `Topology_ZF_11` theory studies relations between the order and various properties of generated topology. For example one can show that if the order topology is connected, then the order is complete (in the sense that for each set bounded from above the set of upper bounds has a minimum). For a given cardinal κ we can consider generalized notion of $\kappa - separability$. Turns out κ-separability is related to (order) density of sets of cardinality κ for order topologies.

Being a topological group imposes additional structure on the topology of the group, in particular its separation properties. In `Topological_Group_ZF_1.thy` theory it is shown that if a topology is T_0, then it must be T_3 , and that the topology in a topological group is always regular.

For a given normal subgroup of a topological group we can define a topology on the quotient group in a natural way. At the end of the `Topological_Group_ZF_2.thy` theory it is shown that such topology on the quotient group makes it a topological group.

The `Topological_Group_ZF_3.thy` theory studies the topologies on subgroups of a topological group. A couple of nice basic properties are shown, like that the closure of a subgroup is a subgroup, closure of a normal subgroup is normal and, a bit more surprising (to me) property that every locally-compact subgroup of a T_0 group is closed.

In `Complex_ZF` we construct complex numbers starting from a complete ordered field (a model of real numbers). We also define the notation for writing about complex numbers and prove that the structure of complex numbers constructed there satisfies the axioms of complex numbers used in Metamath.

`MMI_prelude` defines the `mmisar0` context in which most theorems translated from Metamath are proven. It also contains a chapter explaining how the translation works.

In the `Metamath_interface` theory we prove a theorem that the `mmisar0` context is valid (can be used) in the `complex0` context. All theories using the translated results will import the `Metamath_interface` theory. The `Metamath_sampler` theory provides some examples of using the translated theorems in the `complex0` context.

The theories `MMI_logic_and_sets`, `MMI_Complex`, `MMI_Complex_1` and `MMI_Complex_2` contain the theorems imported from the Metamath's set.mm database. As the translated proofs are rather verbose these theories are not printed in this proof document. The full list of translated facts can be found in the `Metamath_theorems.txt` file included in the IsarMathLib distribution. The `MMI_examples` provides some theorems imported from Metamath that are printed in this proof document as examples of how translated proofs look like.

end

2 First Order Logic

theory `Fol1` **imports** `ZF.Trancl`

begin

Isabelle/ZF builds on the first order logic. Almost everything one would like to have in this area is covered in the standard Isabelle libraries. The material in this theory provides some lemmas that are missing or allow for a more readable proof style.

2.1 Notions and lemmas in FOL

This section contains mostly shortcuts and workarounds that allow to use more readable coding style.

The next lemma serves as a workaround to problems with applying the definition of transitivity (of a relation) in our coding style (any attempt to do something like `using trans_def` puts Isabelle in an infinite loop).

```
lemma Fol1_L2: assumes
  A1: ∀ x y z. ⟨x, y⟩ ∈ r ∧ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
  shows trans(r)
proof -
  from A1 have
    ∀ x y z. ⟨x, y⟩ ∈ r ⟶ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
    using imp_conj by blast
  then show thesis unfolding trans_def by blast
qed
```

Another workaround for the problem of Isabelle simplifier looping when the transitivity definition is used.

```
lemma Fol1_L3: assumes A1: trans(r) and A2: ⟨ a,b⟩ ∈ r  ∧ ⟨ b,c⟩ ∈ r
  shows ⟨ a,c⟩ ∈ r
proof -
  from A1 have  ∀x y z. ⟨x, y⟩ ∈ r ⟶ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
   unfolding trans_def by blast
  with A2 show thesis using imp_conj by fast
qed
```

There is a problem with application of the definition of asymetry for relations. The next lemma is a workaround.

```
lemma Fol1_L4:
  assumes A1: antisym(r) and A2: ⟨ a,b⟩ ∈ r   ⟨ b,a⟩ ∈ r
  shows a=b
proof -
  from A1 have ∀ x y. ⟨ x,y⟩ ∈ r ⟶ ⟨ y,x⟩ ∈ r ⟶ x=y
    unfolding antisym_def by blast
  with A2 show a=b using imp_conj by fast
qed
```

The definition below implements a common idiom that states that (perhaps under some assumptions) exactly one of given three statements is true.

```
definition
  Exactly_1_of_3_holds(p,q,r) ≡
  (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
```

The next lemma allows to prove statements of the form `Exactly_1_of_3_holds(p,q,r)`.

```
lemma Fol1_L5:
  assumes p∨q∨r
  and p ⟶ ¬q ∧ ¬r
  and q ⟶ ¬p ∧ ¬r
  and r ⟶ ¬p ∧ ¬q
  shows Exactly_1_of_3_holds(p,q,r)
proof -
  from assms have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
```

```
    by blast
  then show Exactly_1_of_3_holds (p,q,r)
    unfolding Exactly_1_of_3_holds_def by fast
qed
```

If exactly one of p, q, r holds and p is not true, then q or r.

```
lemma Fol1_L6:
  assumes A1: ¬p and A2: Exactly_1_of_3_holds(p,q,r)
  shows q∨r
proof -
  from A2 have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly_1_of_3_holds_def by fast
  hence p ∨ q ∨ r by blast
  with A1 show q ∨ r by simp
qed
```

If exactly one of p, q, r holds and q is true, then r can not be true.

```
lemma Fol1_L7:
  assumes A1: q and A2: Exactly_1_of_3_holds(p,q,r)
  shows ¬r
proof -
   from A2 have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly_1_of_3_holds_def by fast
  with A1 show ¬r by blast
qed
```

The next lemma demonstrates an elegant form of the `Exactly_1_of_3_holds(p,q,r)` predicate.

```
lemma Fol1_L8:
  shows Exactly_1_of_3_holds(p,q,r) ⟷ (p⟷q⟷r) ∧ ¬(p∧q∧r)
proof
  assume Exactly_1_of_3_holds(p,q,r)
  then have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    unfolding Exactly_1_of_3_holds_def by fast
  thus (p⟷q⟷r) ∧ ¬(p∧q∧r) by blast
next assume (p⟷q⟷r) ∧ ¬(p∧q∧r)
  hence
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    by auto
  then show Exactly_1_of_3_holds(p,q,r)
    unfolding Exactly_1_of_3_holds_def by fast
qed
```

A property of the `Exactly_1_of_3_holds` predicate.

```
lemma Fol1_L8A: assumes A1: Exactly_1_of_3_holds(p,q,r)
```

```
  shows p ⟷ ¬(q ∨ r)
proof -
  from A1 have (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶
¬p ∧ ¬q)
    unfolding Exactly_1_of_3_holds_def by fast
  then show p ⟷ ¬(q ∨ r) by blast
qed
```

Exclusive or definition. There is one also defined in the standard Isabelle, denoted `xor`, but it relates to boolean values, which are sets. Here we define a logical functor.

```
definition
  Xor (infixl Xor 66) where
  p Xor q ≡ (p∨q) ∧ ¬(p ∧ q)
```

The "exclusive or" is the same as negation of equivalence.

```
lemma Fol1_L9: shows p Xor q ⟷ ¬(p⟷q)
  using Xor_def by auto
```

Equivalence relations are symmetric.

```
lemma equiv_is_sym: assumes A1: equiv(X,r) and A2: ⟨x,y⟩ ∈ r
  shows ⟨y,x⟩ ∈ r
proof -
  from A1 have sym(r) using equiv_def by simp
  then have ∀x y. ⟨x,y⟩ ∈ r ⟶ ⟨y,x⟩ ∈ r
    unfolding sym_def by fast
  with A2 show ⟨y,x⟩ ∈ r by blast
qed


end
```

3 ZF set theory basics

```
theory ZF1 imports ZF.Perm

begin
```

The standard Isabelle distribution contains lots of facts about basic set theory. This theory file adds some more.

3.1 Lemmas in Zermelo-Fraenkel set theory

Here we put lemmas from the set theory that we could not find in the standard Isabelle distribution or just so that they are easier to find.

In Isabelle/ZF the set difference is written with a minus sign $A - B$ because the standard backslash character is reserved for other purposes. The next abbreviation declares that we want the set difference character $A \setminus B$ to be synonymous with the minus sign.

```
abbreviation set_difference (infixl \ 65) where A\B ≡ A-B
```

Complement of the complement is the set.

```
lemma diff_diff_eq: assumes A⊆X shows X\(X\A) = A using assms by auto
```

A set cannot be a member of itself. This is exactly lemma `mem_not_refl` from Isabelle/ZF `upair.thy`, we put it here for easy reference.

```
lemma mem_self: shows x∉x by (rule mem_not_refl)
```

If we remove an element and put it back we get the set back.

```
lemma rem_add_eq: assumes a∈A shows (A\{a}) ∪ {a} = A
  using assms by auto
```

Applying a transformation to equal values yields equal results.. It is surprising but we do have to use this as a rule in rare cases.

```
lemma same_constr: assumes x=y shows P(x) = P(y)
  using assms by simp
```

If one collection is contained in another, then we can say the same about their unions.

```
lemma collection_contain: assumes A⊆B shows ⋃A ⊆ ⋃B
proof
  fix x assume x ∈ ⋃A
  then obtain X where x∈X and X∈A by auto
  with assms show x ∈ ⋃B by auto
qed
```

In ZF set theory the zero of natural numbers is the same as the empty set. In the next abbreviation we declare that we want 0 and $\emptyset$ to be synonyms so that we can use $\emptyset$ instead of 0 when appropriate.

```
abbreviation empty_set (∅) where ∅ ≡ 0
```

If all sets of a nonempty collection are the same, then its union is the same.

```
lemma ZF1_1_L1: assumes C≠∅ and ∀y∈C. b(y) = A
  shows (⋃y∈C. b(y)) = A using assms by blast
```

The union af all values of a constant meta-function belongs to the same set as the constant.

```
lemma ZF1_1_L2: assumes A1:C≠∅ and A2: ∀x∈C. b(x) ∈ A
  and A3: ∀x y. x∈C ∧ y∈C ⟶ b(x) = b(y)
  shows (⋃x∈C. b(x))∈A
proof -
  from A1 obtain x where D1: x∈C by auto
  with A3 have ∀y∈C. b(y) = b(x) by blast
```

```
  with A1 have (⋃y∈C. b(y)) = b(x)
    using ZF1_1_L1 by simp
  with D1 A2 show thesis by simp
qed
```

If two meta-functions are the same on a cartesian product, then the subsets defined by them are the same. I am surprised Isabelle can not handle this automatically.

```
lemma ZF1_1_L4: assumes A1: ∀x∈X.∀y∈Y. a(x,y) = b(x,y)
  shows {a(x,y). ⟨x,y⟩ ∈ X×Y} = {b(x,y). ⟨x,y⟩ ∈ X×Y}
proof
  show {a(x, y). ⟨x,y⟩ ∈ X × Y} ⊆ {b(x, y). ⟨x,y⟩ ∈ X × Y}
  proof
    fix z assume z ∈ {a(x, y) . ⟨x,y⟩ ∈ X × Y}
    with A1 show  z ∈ {b(x,y).⟨x,y⟩ ∈ X×Y} by auto
  qed
  show {b(x, y). ⟨x,y⟩ ∈ X × Y} ⊆ {a(x, y). ⟨x,y⟩ ∈ X × Y}
  proof
    fix z assume z ∈ {b(x, y). ⟨x,y⟩ ∈ X × Y}
    with A1 show z ∈ {a(x,y).⟨x,y⟩ ∈ X×Y} by auto
  qed
qed
```

If two meta-functions are the same on a cartesian product, then the subsets defined by them are the same. This is similar to `ZF1_1_L4`, except that the set definition varies over `p∈X×Y` rather than `⟨ x,y⟩∈X×Y`.

```
lemma ZF1_1_L4A: assumes A1: ∀x∈X.∀y∈Y. a(⟨ x,y⟩) = b(x,y)
  shows {a(p). p ∈ X×Y} = {b(x,y). ⟨x,y⟩ ∈ X×Y}
proof
  { fix z assume z ∈ {a(p). p∈X×Y}
    then obtain p where D1: z=a(p) p∈X×Y by auto
    let x = fst(p) let y = snd(p)
    from A1 D1 have z ∈ {b(x,y). ⟨x,y⟩ ∈ X×Y} by auto
  } then show {a(p). p ∈ X×Y} ⊆ {b(x,y). ⟨x,y⟩ ∈ X×Y} by blast
next
  { fix z assume z ∈ {b(x,y). ⟨x,y⟩ ∈ X×Y}
    then obtain x y where D1: ⟨x,y⟩ ∈ X×Y z=b(x,y) by auto
    let p = ⟨ x,y⟩
    from A1 D1 have p∈X×Y z = a(p) by auto
    then have z ∈ {a(p). p ∈ X×Y} by auto
  } then show {b(x,y). ⟨x,y⟩ ∈ X×Y} ⊆ {a(p). p ∈ X×Y} by blast
qed
```

A lemma about inclusion in cartesian products. Included here to remember that we need the $U \times V \neq \emptyset$ assumption.

```
lemma prod_subset: assumes U×V≠∅ U×V ⊆ X×Y shows U⊆X and V⊆Y
  using assms by auto
```

A technical lemma about sections in cartesian products.

```
lemma section_proj: assumes A ⊆ X×Y and U×V ⊆ A and x ∈ U  y ∈ V
  shows U ⊆ {t∈X. ⟨t,y⟩ ∈ A} and V ⊆ {t∈Y. ⟨x,t⟩ ∈ A}
  using assms by auto
```

If two meta-functions are the same on a set, then they define the same set by separation.

```
lemma ZF1_1_L4B: assumes ∀x∈X. a(x) = b(x)
  shows {a(x). x∈X} = {b(x). x∈X}
  using assms by simp
```

A set defined by a constant meta-function is a singleton.

```
lemma ZF1_1_L5: assumes X≠∅ and ∀x∈X. b(x) = c
  shows {b(x). x∈X} = {c} using assms by blast
```

Most of the time, `auto` does this job, but there are strange cases when the next lemma is needed.

```
lemma subset_with_property: assumes Y = {x∈X. b(x)}
  shows Y ⊆ X
  using assms by auto
```

If set A is contained in set B and exist elements x, y of the set A that satisfy a predicate then exist elements of the set B that satisfy the predicate.

```
lemma exist2_subset: assumes A⊆B and ∃x∈A. ∃y∈A. φ(x,y)
  shows ∃x∈B. ∃y∈B. φ(x,y)
  using assms by blast
```

We can choose an element from a nonempty set.

```
lemma nonempty_has_element: assumes X≠∅ shows ∃x. x∈X
  using assms by auto
```

In Isabelle/ZF the intersection of an empty family is empty. This is exactly lemma `Inter_0` from Isabelle's `equalities` theory. We repeat this lemma here as it is very difficult to find. This is one reason we need comments before every theorem: so that we can search for keywords.

```
lemma inter_empty_empty: shows ⋂∅ = ∅ by (rule Inter_0)
```

If an intersection of a collection is not empty, then the collection is not empty. We are (ab)using the fact the the intersection of empty collection is defined to be empty.

```
lemma inter_nempty_nempty: assumes ⋂A ≠ ∅ shows A≠∅
  using assms by auto
```

For two collections S, T of sets we define the product collection as the collections of cartesian products $A \times B$, where $A \in S, B \in T$.

```
definition
  ProductCollection(T,S) ≡ ⋃U∈T.{U×V. V∈S}
```

The union of the product collection of collections S, T is the cartesian product of $\bigcup S$ and $\bigcup T$.

```
lemma ZF1_1_L6: shows ⋃ ProductCollection(S,T) = ⋃S × ⋃T
  using ProductCollection_def by auto
```

An intersection of subsets is a subset.

```
lemma inter_subsets_subset: assumes ∀i∈I. P(i) ⊆ X
  shows (⋂i∈I. P(i)) ⊆ X
proof -
  { assume I=∅
    hence thesis by simp
  }
  moreover
  { assume I≠∅
    with assms have thesis by force
  }
  ultimately show thesis by blast
qed
```

Intersection of a collection of subsets of X is a subset of X. Similar to `inter_subsets_subset` but for non-indexed collections.

```
lemma inter_subsets_subset1: assumes M⊆Pow(X) shows ⋂M ⊆ X
proof -
  { assume M≠∅
    with assms have ⋂M ⊆ X by blast
  }
  moreover
  { assume M=∅
    hence ⋂M ⊆ X by simp
  }
  ultimately show thesis by blast
qed
```

Intersection of a smaller (but nonempty) collection of sets is larger. Note the assumption that the smaller collection is nonepty in necessary here.

```
lemma inter_index_mono: assumes I⊆M I≠∅
  shows (⋂i∈M. P(i)) ⊆ (⋂i∈I. P(i))
  using assms by auto
```

Isabelle/ZF has a "THE" construct that allows to define an element if there is only one such that is satisfies given predicate. In pure ZF we can express something similar using the indentity proven below.

```
lemma ZF1_1_L8: shows ⋃{x} = x by auto
```

Some properties of singletons.

```
lemma ZF1_1_L9: assumes ∃! x. x∈A ∧ φ(x)
  shows
```

```
  ∃a. {x∈A. φ(x)} = {a}
  ⋃ {x∈A. φ(x)} ∈ A
  φ(⋃ {x∈A. φ(x)})
  ∃x∈A. φ(x)
proof -
  from assms(1) show ∃a. {x∈A. φ(x)} = {a} by auto
  then obtain a where I: {x∈A. φ(x)} = {a} by auto
  then have ⋃ {x∈A. φ(x)} = a by auto
  moreover
  from I have a ∈ {x∈A. φ(x)} by simp
  hence a∈A and φ(a) by auto
  ultimately show ⋃ {x∈A. φ(x)} ∈ A and φ(⋃ {x∈A. φ(x)})
    by auto
  from assms show ∃x∈A. φ(x) by auto
qed
```

A simple version of ZF1_1_L9.

```
corollary singleton_extract: assumes ∃! x. x∈A
  shows (⋃ A) ∈ A
proof -
  from assms have ∃! x. x∈A ∧ True by simp
  then have ⋃ {x∈A. True} ∈ A by (rule ZF1_1_L9)
  thus (⋃ A) ∈ A by simp
qed
```

A criterion for when a set defined by comprehension is a singleton.

```
lemma singleton_comprehension:
  assumes A1: y∈X and A2: ∀x∈X. ∀y∈X. P(x) = P(y)
  shows (⋃{P(x). x∈X}) = P(y)
proof -
  let A = {P(x). x∈X}
  have ∃! c. c ∈ A
  proof
    from A1 show ∃c. c ∈ A by auto
  next
    fix a b assume a ∈ A and b ∈ A
    then obtain x t where
      x ∈ X a = P(x) and t ∈ X b = P(t)
      by auto
    with A2 show a=b by blast
  qed
  then have (⋃A) ∈ A by (rule singleton_extract)
  then obtain x where x ∈ X and (⋃A) = P(x)
    by auto
  from A1 A2 ‹x ∈ X› have P(x) = P(y)
    by blast
  with ‹(⋃A) = P(x)› show (⋃A) = P(y) by simp
qed
```

Adding an element of a set to that set does not change the set.

```
lemma set_elem_add: assumes x∈X shows X ∪ {x} = X using assms
  by auto
```

Here we define a restriction of a collection of sets to a given set. In romantic math this is typically denoted $X \cap M$ and means $\{X \cap A : A \in M\}$. Note there is also restrict(f, A) defined for relations in ZF.thy.

```
definition
  RestrictedTo (infixl {restricted to} 70) where
  M {restricted to} X ≡ {X ∩ A . A ∈ M}
```

A lemma on a union of a restriction of a collection to a set.

```
lemma union_restrict:
  shows ⋃(M {restricted to} X) = (⋃M) ∩ X
  using RestrictedTo_def by auto
```

Next we show a technical identity that is used to prove sufficiency of some condition for a collection of sets to be a base for a topology.

```
lemma ZF1_1_L10: assumes A1: ∀U∈C. ∃A∈B. U = ⋃A
  shows ⋃⋃ {⋃{A∈B. U = ⋃A}. U∈C} = ⋃C
proof
  show ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A}) ⊆ ⋃C by blast
  show ⋃C ⊆ ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A})
  proof
    fix x assume x ∈ ⋃C
    show x ∈ ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A})
    proof -
      from ‹x ∈ ⋃C› obtain U where U∈C ∧ x∈U by auto
      with A1 obtain A where A∈B ∧ U = ⋃A by auto
      from ‹U∈C ∧ x∈U› ‹A∈B ∧ U = ⋃A› show x∈ ⋃(⋃U∈C. ⋃{A ∈ B .
U = ⋃A})
 by auto
    qed
  qed
qed
```

Standard Isabelle uses a notion of `cons(A,a)` that can be thought of as $A \cup \{a\}$.

```
lemma consdef: shows cons(a,A) = A ∪ {a}
  using cons_def by auto
```

If a difference between a set and a singleton is empty, then the set is empty or it is equal to the singleton.

```
lemma singl_diff_empty: assumes A\{x} = ∅
  shows A = ∅ ∨ A = {x}
  using assms by auto
```

If a difference between a set and a singleton is the set, then the only element of the singleton is not in the set.

```
lemma singl_diff_eq: assumes A1: A\{x} = A
  shows x ∉ A
proof -
  have x ∉ A - {x} by auto
  with A1 show x ∉ A by simp
qed
```

Simple substitution in membership, has to be used by rule in very rare cases.

```
lemma eq_mem: assumes x∈A and y=x shows y∈A
  using assms by simp
```

A basic property of sets defined by comprehension.

```
lemma comprehension: assumes a ∈ {x∈X. p(x)}
  shows a∈X and p(a) using assms by auto
```

A basic property of a set defined by another type of comprehension.

```
lemma comprehension_repl: assumes y ∈ {p(x). x∈X}
  shows ∃x∈X. y = p(x) using assms by auto
```

The inverse of the `comprehension` lemma.

```
lemma mem_cond_in_set: assumes φ(c) and c∈X
  shows c ∈ {x∈X. φ(x)} using assms by blast
```

The image of a set by a greater relation is greater.

```
lemma image_rel_mono: assumes r⊆s shows r(A) ⊆ s(A)
  using assms by auto
```

A technical lemma about relations: if x is in its image by a relation U and that image is contained in some set C, then the image of the singleton $\{x\}$ by the relation $U \cup C \times C$ equals C.

```
lemma image_greater_rel:
  assumes  x ∈ U{x}  and U{x} ⊆ C
  shows (U ∪ C×C){x} = C
  using assms image_Un_left by blast
```

Reformulation of the definition of composition of two relations:

```
lemma rel_compdef:
  shows ⟨x,z⟩ ∈ r O s  ⟷  (∃y. ⟨x,y⟩ ∈ s ∧ ⟨y,z⟩ ∈ r)
  unfolding comp_def by auto
```

Domain and range of the relation of the form $\bigcup\{U \times U : U \in P\}$ is $\bigcup P$:

```
lemma domain_range_sym: shows domain(⋃{U×U. U∈P}) = ⋃P and range(⋃{U×U.
U∈P}) = ⋃P
  by auto
```

The product of converses is equal to in converse of a product

```
lemma prod_converse: assumes M ⊆ Pow(X×X)
```

```
  shows ⋂{converse(A). A∈M} = converse(⋂M)
proof -
  { assume M≠∅
    from assms(1) have converse(⋂M) ⊆ X×X
    using inter_subsets_subset1 by auto
    let N = ⋂{converse(A). A∈M}
    from assms(1) have ∀A∈M. converse(A) ⊆ X×X by auto
    then have N ⊆ X×X using inter_subsets_subset by simp
    { fix p assume p∈N
      with ‹N ⊆ X×X› obtain x y where p=⟨x,y⟩ by blast
      with ‹M≠∅› ‹p∈N› have p ∈ converse(⋂M) by auto
    } hence N ⊆ converse(⋂M) by auto
    moreover
    { fix p assume p∈converse(⋂M)
      with ‹converse(⋂M) ⊆ X×X› obtain x y where p=⟨x,y⟩ by blast
      with ‹M≠∅› ‹p∈converse(⋂M)› have p∈N by auto
    } hence converse(⋂M) ⊆ N by auto
    ultimately have thesis by auto
  }
  moreover
  { assume M=∅
    then have thesis by simp
  }
  ultimately show thesis by blast
qed
```

An identity for the square (in the sense of composition) of a symmetric relation.

```
lemma symm_sq_prod_image: assumes converse(r) = r
  shows r O r = ⋃{(r{x})×(r{x}). x ∈ domain(r)}
proof
  { fix p assume p ∈ r O r
    then obtain y z where ⟨y,z⟩ = p by auto
    with ‹p ∈ r O r› obtain x where ⟨y,x⟩ ∈ r and ⟨x,z⟩ ∈ r
      using rel_compdef by auto
    from ‹⟨y,x⟩ ∈ r› have ⟨x,y⟩ ∈ converse(r) by simp
    with assms ‹⟨x,z⟩ ∈ r› ‹⟨y,z⟩ = p› have ∃x∈domain(r). p ∈ (r{x})×(r{x})
      by auto
  } thus r O r ⊆ (⋃{(r{x})×(r{x}). x ∈ domain(r)})
    by blast
  { fix x assume x ∈ domain(r)
    have (r{x})×(r{x}) ⊆ r O r
    proof -
      { fix p assume p ∈ (r{x})×(r{x})
        then obtain y z where ⟨y,z⟩ = p y ∈ r{x} z ∈ r{x}
          by auto
        from ‹y ∈ r{x}› have ⟨x,y⟩ ∈ r by auto
        then have ⟨y,x⟩ ∈ converse(r) by simp
        with assms ‹z ∈ r{x}› ‹⟨y,z⟩ = p› have p ∈ r O r by auto
```

```
    } thus thesis  by auto
  qed
} thus (⋃{(r{x})×(r{x}). x ∈ domain(r)}) ⊆ r O r
  by blast
qed
```

Square of a reflexive relation contains the relation. Recall that in ZF the identity function on X is the same as the diagonal of $X \times X$, i.e. $id(X) = \{\langle x, x\rangle : x \in X\}$.

```
lemma refl_square_greater: assumes r ⊆ X×X id(X) ⊆ r
  shows r ⊆ r O r using assms by auto
```

A reflexive relation is contained in the union of products of its singleton images.

```
lemma refl_union_singl_image:
  assumes A ⊆ X×X and id(X)⊆A shows A ⊆ ⋃{A{x}×A{x}. x ∈ X}
proof -
  { fix p assume p∈A
    with assms(1) obtain x y where x∈X y∈X and p=⟨x,y⟩ by auto
    with assms(2) ‹p∈A› have ∃x∈X. p ∈ A{x}×A{x} by auto
  } thus thesis by auto
qed
```

If the cartesian product of the images of x and y by a symmetric relation W has a nonempty intersection with R then x is in relation $W \circ (R \circ W)$ with y.

```
lemma sym_rel_comp:
  assumes W=converse(W) and (W{x})×(W{y}) ∩ R ≠ ∅
  shows ⟨x,y⟩ ∈ (W O (R O W))
proof -
  from assms(2) obtain s t where s∈W{x} t∈W{y} and ⟨s,t⟩∈R
    by blast
  then have ⟨x,s⟩ ∈ W and ⟨y,t⟩ ∈ W by auto
  from ‹⟨x,s⟩ ∈ W› ‹⟨s,t⟩ ∈ R› have ⟨x,t⟩ ∈ R O W by auto
  from ‹⟨y,t⟩ ∈ W› have ⟨t,y⟩ ∈ converse(W) by blast
  with assms(1) ‹⟨x,t⟩ ∈ R O W› show thesis by auto
qed
```

Suppose we have two families of sets $\{A(i)\}_{i\in I}$ and $\{B(i)\}_{i\in I}$, indexed by a nonepty set of indices I and such that for every index $i \in I$ we have inclusion $A(i) \circ A(i) \subseteq B(i)$. Then a similar inclusion holds for the products of the families, namely $(\bigcap_{i\in I} A(i)) \circ (\bigcap_{i\in I} A(i)) \subseteq (\bigcap_{i\in I} B(i)$.

```
lemma square_incl_product: assumes I≠∅ ∀i∈I. A(i) O A(i) ⊆ B(i)
  shows (⋂i∈I. A(i)) O (⋂i∈I. A(i)) ⊆ (⋂i∈I. B(i))
  using assms by force
```

It's hard to believe but there are cases where we have to reference this rule.

lemma set_mem_eq: **assumes** x∈A A=B **shows** x∈B **using** assms **by** simp

Given some family $\mathcal{A}$ of subsets of X we can define the family of supersets of $\mathcal{A}$.

definition
  Supersets(X,𝒜) ≡ {B∈Pow(X). ∃A∈𝒜. A⊆B}

If A is a member of a collection of sets $\mathcal{A}$ then it is one of supersets of $\mathcal{A}$.

lemma superset_gen: **assumes** A⊆X A∈𝒜 **shows** A ∈ Supersets(X,𝒜)
  using assms **unfolding** Supersets_def **by** auto

The whole space is a superset of any nonempty collection of its subsets.

lemma space_superset: **assumes** 𝒜≠∅ 𝒜⊆Pow(X) **shows** X ∈ Supersets(X,𝒜)
proof -
  from assms(1) **obtain** A **where** A∈𝒜 **by** auto
  with assms(2) **show** thesis **unfolding** Supersets_def **by** auto
qed

The collection of supersets of an empty set is empty. In particular the whole space X is not a superset of an empty set.

lemma supersets_of_empty: **shows** Supersets(X,∅) = ∅
  unfolding Supersets_def **by** auto

However, when the space is empty the collection of supersets does not have to be empty - the collection of supersets of the singleton collection containing only the empty set is this collection.

lemma supersets_in_empty: **shows** Supersets(∅,{∅}) = {∅}
  unfolding Supersets_def **by** auto

This can be done by the auto method, but sometimes takes a long time.

lemma witness_exists: **assumes** x∈X **and** φ(x) **shows** ∃x∈X. φ(x)
  using assms **by** auto

Another lemma that concludes existence of some set.

lemma witness_exists1: **assumes** x∈X φ(x) ψ(x)
  shows ∃x∈X. φ(x) ∧ ψ(x)
  using assms **by** auto

The next lemma has to be used as a rule in some rare cases.

lemma exists_in_set: **assumes** ∀x. x∈A ⟶ φ(x) **shows** ∀x∈A. φ(x)
  using assms **by** simp

If x belongs to a set where a property holds, then the property holds for x. This has to be used as rule in rare cases.

lemma property_holds: **assumes** ∀t∈X. φ(t) **and** x∈X
  shows φ(x) **using** assms **by** simp

Set comprehensions defined by equal expressions are the equal. The second assertion is actually about functions, which are sets of pairs as illustrated in lemma `fun_is_set_of_pairs` in `func1.thy`

```
lemma set_comp_eq: assumes ∀x∈X. p(x) = q(x)
  shows {p(x). x∈X} = {q(x). x∈X} and {⟨x,p(x)⟩. x∈X} = {⟨x,q(x)⟩. x∈X}
  using assms by auto
```

If every element of a non-empty set $X \subseteq Y$ satisfies a condition then the set of elements of Y that satisfy the condition is non-empty.

```
lemma non_empty_cond: assumes X≠∅ X⊆Y and ∀x∈X. P(x)
  shows {x∈Y. P(x)} ≠ 0 using assms by auto
```

If z is a pair, then the cartesian product of the singletons of its elements is the same as the singleton $\{z\}$.

```
lemma pair_prod: assumes z = ⟨x,y⟩ shows {x}×{y} = {z}
  using assms by blast
```

end

4 Natural numbers in IsarMathLib

theory `Nat_ZF_IML` **imports** `ZF.ArithSimp`

begin

The ZF set theory constructs natural numbers from the empty set and the notion of a one-element set. Namely, zero of natural numbers is defined as the empty set. For each natural number n the next natural number is defined as $n \cup \{n\}$. With this definition for every non-zero natural number we get the identity $n = \{0, 1, 2, .., n-1\}$. It is good to remember that when we see an expression like $f : n \to X$. Also, with this definition the relation "less or equal than" becomes "$\subseteq$" and the relation "less than" becomes "$\in$".

4.1 Induction

The induction lemmas in the standard Isabelle's Nat.thy file like for example `nat_induct` require the induction step to be a higher order statement (the one that uses the $\Longrightarrow$ sign). I found it difficult to apply from Isar, which is perhaps more of an indication of my Isar skills than anything else. Anyway, here we provide a first order version that is easier to reference in Isar declarative style proofs.

The next theorem is a version of induction on natural numbers that I was thought in school.

```
theorem ind_on_nat:
  assumes A1: n∈nat and A2: P(0) and A3: ∀k∈nat. P(k)⟶P(succ(k))
  shows P(n)
proof -
  note A1 A2
  moreover
  { fix x
    assume x∈nat  P(x)
    with A3 have P(succ(x)) by simp }
  ultimately show  P(n) by (rule nat_induct)
qed
```

A nonzero natural number has a predecessor.

```
lemma Nat_ZF_1_L3: assumes A1: n ∈ nat and A2: n≠0
  shows ∃k∈nat. n = succ(k)
proof -
  from A1 have n ∈ {0} ∪ {succ(k). k∈nat}
    using nat_unfold by simp
  with A2 show thesis by simp
qed
```

What is succ, anyway? It's a union with the singleton of the set.

```
lemma succ_explained: shows succ(n) = n ∪ {n}
  using succ_iff by auto
```

The singleton containing the empty set is a natural number.

```
lemma one_is_nat: shows {0} ∈ nat {0} = succ(0) {0} = 1
proof -
  show {0} = succ(0) using succ_explained by simp
  then show {0} ∈ nat by simp
  show {0}=1 by blast
qed
```

If k is a member of $succ(n)$ but is not n, then it must be the member of n.

```
lemma mem_succ_not_eq: assumes k∈succ(n) k≠n
  shows k∈n using assms succ_explained by simp
```

Empty set is an element of every natural number which is not zero.

```
lemma empty_in_every_succ: assumes A1: n ∈ nat
  shows 0 ∈ succ(n)
proof -
  note A1
  moreover have 0 ∈ succ(0) by simp
  moreover
  { fix k assume k ∈ nat and A2: 0 ∈ succ(k)
    then have succ(k) ⊆ succ(succ(k)) by auto
    with A2 have 0 ∈ succ(succ(k)) by auto
  } then have ∀k ∈ nat. 0 ∈ succ(k) ⟶ 0 ∈ succ(succ(k))
```

```
    by simp
  ultimately show 0 ∈ succ(n) by (rule ind_on_nat)
qed
```

Various forms of saying that for natural numbers taking the successor is the same as adding one.

```
lemma succ_add_one: assumes n∈nat
  shows
    n #+ 1 = succ(n)
    n #+ 1 ∈ nat
    {0} #+ n = succ(n)
    n #+ {0} = succ(n)
    succ(n) ∈ nat
    0 ∈ n #+ 1
    n ⊆ n #+ 1
proof -
  from assms show n #+ 1 = succ(n) n #+ 1 ∈ nat succ(n) ∈ nat by simp_all
  moreover from assms have {0} = 1 and n #+ 1 = 1 #+ n by auto
  ultimately show {0} #+ n = succ(n) and n #+ {0} = succ(n)
    by simp_all
  from assms ‹n #+ 1 = succ(n)› show 0 ∈ n #+ 1 using empty_in_every_succ
    by simp
  from assms ‹n #+ 1 = succ(n)› show n ⊆ n #+ 1 using succ_explained
    by auto
qed
```

A more direct way of stating that empty set is an element of every non-zero natural number:

```
lemma empty_in_non_empty: assumes n∈nat n≠0
  shows 0∈n
  using assms Nat_ZF_1_L3 empty_in_every_succ by auto
```

If one natural number is less than another then their successors are in the same relation.

```
lemma succ_ineq: assumes A1: n ∈ nat
  shows ∀i ∈ n. succ(i) ∈ succ(n)
proof -
  note A1
  moreover have ∀k ∈ 0. succ(k) ∈ succ(0) by simp
  moreover
  { fix k assume A2: ∀i∈k. succ(i) ∈ succ(k)
    { fix i assume i ∈ succ(k)
      then have i ∈ k ∨ i = k by auto
      moreover
      { assume i∈k
 with A2 have succ(i) ∈ succ(k) by simp
 hence succ(i) ∈ succ(succ(k)) by auto }
      moreover
```

```
      { assume i = k
 then have succ(i) ∈ succ(succ(k)) by auto }
      ultimately have succ(i) ∈ succ(succ(k)) by auto
    } then have ∀i ∈ succ(k). succ(i) ∈ succ(succ(k))
      by simp
  } then have ∀k ∈ nat.
      ( (∀i∈k. succ(i) ∈ succ(k)) ⟶ (∀i ∈ succ(k). succ(i) ∈ succ(succ(k)))
)
    by simp
  ultimately show ∀i ∈ n. succ(i) ∈ succ(n) by (rule ind_on_nat)
qed
```

For natural numbers if $k \subseteq n$ the similar holds for their successors.

```
lemma succ_subset: assumes A1: k ∈ nat  n ∈ nat and A2: k⊆n
  shows succ(k) ⊆ succ(n)
proof -
  from A1 have T: Ord(k) and Ord(n)
    using nat_into_Ord by auto
  with A2 have succ(k) ≤ succ(n)
    using subset_imp_le by simp
  then show succ(k) ⊆ succ(n) using le_imp_subset
    by simp
qed
```

For any two natural numbers one of them is contained in the other.

```
lemma nat_incl_total: assumes A1: i ∈ nat  j ∈ nat
  shows i ⊆ j ∨ j ⊆ i
proof -
  from A1 have T: Ord(i)  Ord(j)
    using nat_into_Ord by auto
  then have i∈j ∨ i=j ∨ j∈i using Ord_linear
    by simp
  moreover
  { assume i∈j
    with T have i⊆j ∨ j⊆i
      using lt_def leI le_imp_subset by simp }
  moreover
  { assume i=j
    then have i⊆j ∨ j⊆i by simp }
  moreover
  { assume j∈i
    with T have i⊆j ∨ j⊆i
      using lt_def leI  le_imp_subset by simp }
  ultimately show i ⊆ j ∨ j ⊆ i by auto
qed
```

The set of natural numbers is the union of all successors of natural numbers.

```
lemma nat_union_succ: shows nat = (⋃n ∈ nat. succ(n))
proof
```

```
  show nat ⊆ (⋃n ∈ nat. succ(n)) by auto
next
  { fix k assume A2: k ∈ (⋃n ∈ nat. succ(n))
    then obtain n where T: n ∈ nat and I: k ∈ succ(n)
      by auto
    then have k ≤ n using nat_into_Ord lt_def
      by simp
    with T have k ∈ nat using le_in_nat by simp
  } then show  (⋃n ∈ nat. succ(n)) ⊆ nat by auto
qed
```

Successors of natural numbers are subsets of the set of natural numbers.

```
lemma succnat_subset_nat: assumes A1: n ∈ nat shows succ(n) ⊆ nat
proof -
  from A1 have succ(n) ⊆ (⋃n ∈ nat. succ(n)) by auto
  then show succ(n) ⊆ nat using nat_union_succ by simp
qed
```

Element k of a natural number n is a natural number that is smaller than n.

```
lemma elem_nat_is_nat: assumes A1: n ∈ nat  and A2: k∈n
  shows k < n  k ∈ nat  k ≤ n  ⟨k,n⟩ ∈ Le
proof -
  from A1 A2 show k < n using nat_into_Ord lt_def by simp
  with A1 show k ∈ nat using lt_nat_in_nat by simp
  from ‹k < n› show k ≤ n using leI by simp
  with A1 ‹k ∈ nat› show ⟨k,n⟩ ∈ Le using Le_def
    by simp
qed
```

A version of `succ_ineq` without a quantifier, with additional assertion using the `n #+ 1` notation.

```
lemma succ_ineq1:  assumes n ∈ nat i∈n
  shows succ(i) ∈ succ(n) i #+ 1 ∈ n #+ 1 i ∈ n #+ 1
  using assms succ_ineq succ_add_one(1,7) elem_nat_is_nat(2)
  by auto
```

For natural numbers membership and inequality are the same and $k \leq n$ is the same as $k \in \mathrm{succ}(n)$. The proof relies on lemmas in the standard Isabelle's `Nat` and `Ordinal` theories.

```
lemma nat_mem_lt: assumes n∈nat
  shows k<n ⟷ k∈n and k≤n ⟷ k ∈ succ(n)
  using assms nat_into_Ord Ord_mem_iff_lt by auto
```

If n is a natural number and $k \leq n$, then k is a natural number.

```
lemma leq_nat_is_nat: assumes n∈nat k≤n shows k∈nat
  using assms nat_mem_lt elem_nat_is_nat(2) by auto
```

The term $k \leq n$ is the same as $k < \text{succ}(n)$.

```
lemma leq_mem_succ: shows k≤n ⟷ k < succ(n) by simp
```

If the successor of a natural number k is an element of the successor of n then a similar relations holds for the numbers themselves.

```
lemma succ_mem:
  assumes n ∈ nat succ(k) ∈ succ(n)
  shows k∈n
  using assms elem_nat_is_nat(1) succ_leE nat_into_Ord
    unfolding lt_def by blast
```

The set of natural numbers is the union of its elements.

```
lemma nat_union_nat: shows nat = ⋃ nat
  using elem_nat_is_nat by blast
```

A natural number is a subset of the set of natural numbers.

```
lemma nat_subset_nat: assumes A1: n ∈ nat shows n ⊆ nat
proof -
  from A1 have n ⊆ ⋃ nat by auto
  then show n ⊆ nat using nat_union_nat by simp
qed
```

Adding natural numbers does not decrease what we add to.

```
lemma add_nat_le: assumes A1: n ∈ nat  and A2: k ∈ nat
  shows
  n ≤ n #+ k
  n ⊆ n #+ k
  n ⊆ k #+ n
proof -
  from A1 A2 have n ≤ n  0 ≤ k  n ∈ nat  k ∈ nat
    using nat_le_refl nat_0_le by auto
  then have n #+ 0 ≤ n #+ k by (rule add_le_mono)
  with A1 show n ≤ n #+ k using add_0_right by simp
  then show n ⊆ n #+ k using le_imp_subset by simp
  then show n ⊆ k #+ n using add_commute by simp
qed
```

Result of adding an element of k is smaller than of adding k.

```
lemma add_lt_mono:
  assumes k ∈ nat and j∈k
  shows
  (n #+ j) < (n #+ k)
  (n #+ j) ∈ (n #+ k)
proof -
  from assms have j < k using elem_nat_is_nat by blast
  moreover note ‹k ∈ nat›
  ultimately show (n #+ j) < (n #+ k)   (n #+ j) ∈ (n #+ k)
```

```
    using add_lt_mono2 ltD by auto
qed
```

A technical lemma about a decomposition of a sum of two natural numbers: if a number i is from $m + n$ then it is either from m or can be written as a sum of m and a number from n. The proof by induction w.r.t. to m seems to be a bit heavy-handed, but I could not figure out how to do this directly from results from standard Isabelle/ZF.

```
lemma nat_sum_decomp: assumes A1: n ∈ nat  and A2: m ∈ nat
  shows ∀i ∈ m #+ n. i ∈ m ∨ (∃j ∈ n. i = m #+ j)
proof -
  note A1
  moreover from A2 have ∀i ∈ m #+ 0. i ∈ m ∨ (∃j ∈ 0. i = m #+ j)
    using add_0_right by simp
  moreover have ∀k∈nat.
    (∀i ∈ m #+ k. i ∈ m ∨ (∃j ∈ k. i = m #+ j)) ⟶
    (∀i ∈ m #+ succ(k). i ∈ m ∨ (∃j ∈ succ(k). i = m #+ j))
  proof -
    { fix k assume A3: k ∈ nat
      { assume A4: ∀i ∈ m #+ k. i ∈ m ∨ (∃j ∈ k. i = m #+ j)
   { fix i assume i ∈  m #+ succ(k)
     then have i ∈ m #+ k ∨ i = m #+ k using add_succ_right
       by auto
     moreover from A4 A3 have
       i ∈ m #+ k ⟶ i ∈ m ∨ (∃j ∈ succ(k). i = m #+ j)
       by auto
     ultimately have i ∈ m ∨ (∃j ∈ succ(k). i = m #+ j)
       by auto
   } then have ∀i ∈ m #+ succ(k). i ∈ m ∨ (∃j ∈ succ(k). i = m #+ j)
     by simp
      } then have (∀i ∈ m #+ k. i ∈ m ∨ (∃j ∈ k. i = m #+ j)) ⟶
   (∀i ∈ m #+ succ(k). i ∈ m ∨ (∃j ∈ succ(k). i = m #+ j))
 by simp
    } then show thesis by simp
  qed
  ultimately show ∀i ∈ m #+ n. i ∈ m ∨ (∃j ∈ n. i = m #+ j)
    by (rule ind_on_nat)
qed
```

A variant of induction useful for finite sequences.

```
lemma fin_nat_ind: assumes A1: n ∈ nat and A2: k ∈ succ(n)
  and A3: P(0) and A4: ∀j∈n. P(j)  ⟶ P(succ(j))
  shows P(k)
proof -
  from A2 have k ∈ n ∨ k=n by auto
  with A1 have k ∈ nat using elem_nat_is_nat by blast
  moreover from A3 have 0 ∈ succ(n) ⟶ P(0) by simp
  moreover from A1 A4 have
    ∀k ∈ nat. (k ∈ succ(n) ⟶ P(k)) ⟶ (succ(k) ∈ succ(n) ⟶ P(succ(k)))
```

```
    using nat_into_Ord Ord_succ_mem_iff by auto
  ultimately have k ∈ succ(n) ⟶ P(k)
    by (rule ind_on_nat)
  with A2 show P(k) by simp
qed
```

Some properties of positive natural numbers.

```
lemma succ_plus: assumes n ∈ nat  k ∈ nat
  shows
  succ(n #+ j) ∈ nat
  succ(n) #+ succ(j) = succ(succ(n #+ j))
  using assms by auto
```

If k is in the successor of n, then the predecessor of k is in n.

```
lemma pred_succ_mem: assumes n∈nat n≠0 k∈succ(n) shows pred(k)∈n
proof -
  from assms(1,3) have k∈nat using succnat_subset_nat by blast
  { assume k≠0
    with ‹k∈nat› obtain j where j∈nat and k=succ(j)
      using Nat_ZF_1_L3 by auto
    with assms(1,3) have pred(k)∈n using succ_mem pred_succ_eq
      by simp
  }
  moreover
  { assume k=0
    with assms(1,2) have pred(k)∈n
      using pred_0 empty_in_non_empty by simp
  } ultimately show thesis by blast
qed
```

For non-zero natural numbers $\text{pred}(n) = n - 1$.

```
lemma pred_minus_one: assumes n∈nat n≠0
  shows n #- 1 = pred(n)
proof -
  from assms obtain k where n=succ(k)
    using Nat_ZF_1_L3 by blast
  with assms show thesis
    using pred_succ_eq eq_succ_imp_eq_m1 by simp
qed
```

For natural numbers if $j \in n$ then $j + 1 \subseteq n$.

```
lemma mem_add_one_subset: assumes n ∈ nat k∈n shows k #+ 1 ⊆ n
proof -
  from assms have k #+ 1 ∈ succ(n)
    using elem_nat_is_nat(2) succ_ineq1 succ_add_one(1) by simp
  with assms(1) show k #+ 1 ⊆ n using nat_mem_lt(2) le_imp_subset
    by blast
qed
```

For a natural n if $k \in n+1$ then $k+1 \leq n+1$.

```
lemma succ_ineq2: assumes n ∈ nat k ∈ n #+ 1
  shows k #+ 1 ≤ n #+ 1 and k≤n
proof -
  from assms show k≤n using succ_add_one(1) nat_mem_lt(2)
    by simp
  with assms(1) show  k #+ 1 ≤ n #+ 1 using add_le_mono1 by blast
qed
```

A nonzero natural number is of the form $n = m+1$ for some natural number m. This is very similar to `Nat_ZF_1_L3` except that we use $n+1$ instead of `succ(n)`.

```
lemma nat_not0_succ: assumes n∈nat n≠0
  shows ∃m∈nat. n = m #+ 1
  using assms Nat_ZF_1_L3 succ_add_one(1) by simp
```

A version of induction on natural numbers that uses the $n+1$ notation instead of `succ(n)`.

```
lemma ind_on_nat1:
  assumes n∈nat and P(0) and ∀k∈nat. P(k)⟶P(k #+ 1)
  shows P(n) using assms succ_add_one(1) ind_on_nat by simp
```

A version of induction for finite sequences using the $n+1$ notation instead of `succ(n)`:

```
lemma fin_nat_ind1:
  assumes n∈nat and P(0) and ∀j∈n. P(j)⟶P(j #+ 1)
  shows ∀k∈n #+ 1. P(k) and P(n)
proof -
  { fix k assume k∈n #+ 1
    with assms have
      n∈nat k∈succ(n) P(0) ∀j∈n. P(j)  ⟶ P(succ(j))
      using succ_add_one(1) elem_nat_is_nat(2) by simp_all
    then have P(k) by (rule fin_nat_ind)
  } thus ∀k∈n #+ 1. P(k) by simp
  with assms(1) show P(n) by simp
qed
```

4.2 Simplification rules for addition and subtraction of natural numbers

This section collects useful simplification rules involving addition and subtraction of natural numbers that we couldn't find in standard Isabelle's `ArithSimp` theory.

Adding and subtracting a natural number cancel each other.

```
lemma add_subtract: assumes m∈nat shows (m #+ n) #- n = m
  using assms diff_add_inverse2 by simp
```

A simplification rule for natural numbers: if $k < n$ then $n-(k+1)+1 = n-k$:

```
lemma nat_subtr_simpl0: assumes n∈nat k∈n
  shows n #- (k #+ 1) #+ 1 = n #- k
proof -
  from assms obtain m where m∈nat and n = m #+ 1
    using nat_not0_succ by blast
  with assms have succ(m) = m #+ 1 succ(m #- k) = m #- k #+ 1
    using elem_nat_is_nat(2) succ_add_one by simp_all
  moreover from assms(2) ‹m∈nat› ‹n = m #+ 1› have
    succ(m) #- k = succ(m #- k)
    using diff_succ succ_ineq2(2) by simp
  ultimately have m  #- k #+ 1 = m #+ 1 #- k by simp
  with ‹n = m #+ 1› show thesis using diff_cancel2 by simp
qed
```

If k is a natural number then $n + k = n + ((n + k)\# - n))$.

```
lemma nat_subtr_simpl1: assumes k∈nat
  shows  n #+ ((n #+ k) #- n) = n #+ k
  using assms diff_add_inverse by simp
```

4.3 Intervals

In this section we consider intervals of natural numbers i.e. sets of the form $\{n + j : j \in 0..k - 1\}$.

The interval is determined by two parameters: starting point and length.

```
definition

  NatInterval(n,k) ≡ {n #+ j. j∈k}
```

Subtracting the beginning af the interval results in a number from the length of the interval. It may sound weird, but note that the length of such interval is a natural number, hence a set.

```
lemma inter_diff_in_len:
  assumes A1: k ∈ nat and A2: i ∈ NatInterval(n,k)
  shows i #- n ∈ k
proof -
  from A2 obtain j where I: i = n #+ j and II: j ∈ k
    using NatInterval_def by auto
  from A1 II have j ∈ nat using elem_nat_is_nat by blast
  moreover from I have i #- n = natify(j) using diff_add_inverse
    by simp
  ultimately have i #- n = j by simp
  with II show thesis by simp
qed
```

Intervals don't overlap with their starting point and the union of an interval with its starting point is the sum of the starting point and the length of the interval.

```
lemma length_start_decomp: assumes  A1: n ∈ nat  k ∈ nat
  shows
  n ∩ NatInterval(n,k) = 0
  n ∪ NatInterval(n,k) = n #+ k
proof -
  { fix i assume A2: i ∈ n and i ∈ NatInterval(n,k)
    then obtain j where I: i = n #+ j and II: j ∈ k
      using NatInterval_def by auto
    from A1 have k ∈ nat using elem_nat_is_nat by blast
    with II have j ∈ nat using elem_nat_is_nat by blast
    with A1 I have n ≤ i using add_nat_le by simp
    moreover from A1 A2 have i < n using elem_nat_is_nat by blast
    ultimately have False using le_imp_not_lt by blast
  } thus n ∩ NatInterval(n,k) = 0 by auto
  from A1 have n ⊆ n #+ k using add_nat_le by simp
  moreover
  { fix i assume i ∈ NatInterval(n,k)
    then obtain j where III: i = n #+ j and IV: j ∈ k
      using NatInterval_def by auto
    with A1 have j < k using elem_nat_is_nat by blast
    with A1 III have i ∈ n #+ k using add_lt_mono2 ltD
      by simp }
  ultimately have n ∪ NatInterval(n,k) ⊆ n #+ k by auto
  moreover from A1 have n #+ k ⊆ n ∪ NatInterval(n,k)
    using nat_sum_decomp NatInterval_def by auto
  ultimately show n ∪ NatInterval(n,k) = n #+ k by auto
qed
```

Some properties of three adjacent intervals.

```
lemma adjacent_intervals3: assumes n ∈ nat  k ∈ nat  m ∈ nat
  shows
  n #+ k #+ m = (n #+ k) ∪ NatInterval(n #+ k,m)
  n #+ k #+ m = n ∪ NatInterval(n,k #+  m)
  n #+ k #+ m = n ∪ NatInterval(n,k) ∪ NatInterval(n #+ k,m)
  using assms add_assoc length_start_decomp by auto

end
```

5 Order relations - introduction

```
theory Order_ZF imports Fol1

begin
```

This theory file considers various notion related to order. We redefine the notions of a preorder, directed set, total order, linear order and partial order to have the same terminology as Wikipedia (I found it very consistent across different areas of math). We also define and study the notions of intervals and bounded sets. We show the inclusion relations between the intervals with endpoints being in certain order. We also show that union of bounded sets are bounded. This allows to show in `Finite_ZF.thy` that finite sets are bounded.

5.1 Definitions

In this section we formulate the definitions related to order relations.

A relation r is "total" on a set X if for all elements a, b of X we have a is in relation with b or b is in relation with a. An example is the $\leq$ relation on numbers.

definition
```
  IsTotal (infixl {is total on} 65) where
  r {is total on} X ≡ (∀a∈X.∀b∈X. ⟨ a,b⟩ ∈ r ∨ ⟨ b,a⟩ ∈ r)
```

A relation r is a partial order on X if it is reflexive on X (i.e. $\langle x, x\rangle$ for every $x \in X$), antisymmetric (if $\langle x, y\rangle \in r$ and $\langle y, x\rangle \in r$, then $x = y$) and transitive $\langle x, y\rangle \in r$ and $\langle y, z\rangle \in r$ implies $\langle x, z\rangle \in r$).

definition
```
  IsPartOrder(X,r) ≡ refl(X,r) ∧ antisym(r) ∧ trans(r)
```

A relation that is reflexive and transitive is called a `preorder`.

definition
```
  IsPreorder(X,r) ≡ refl(X,r) ∧ trans(r)
```

We say that a relation r up-directs a set if every two-element subset of X has an upper bound.

definition
```
  UpDirects (_ {up-directs} _ 90)
  where r {up-directs} X ≡ X≠0 ∧ (∀x∈X.∀y∈X.∃z∈X. ⟨x,z⟩ ∈ r ∧ ⟨y,z⟩
∈ r)
```

Analogously we say that a relation r down-directs a set if every two-element subset of X has a lower bound.

definition
```
  DownDirects (_ {down-directs} _ 90)
  where r {down-directs} X ≡ X≠0 ∧ (∀x∈X.∀y∈X.∃z∈X. ⟨z,x⟩ ∈ r ∧ ⟨z,y⟩
∈ r)
```

Typically the notion that is actually defined is the notion of a `directed set`. or an `upward directed set`, rather than r down-directs X (or r up-directs X). This is a nonempty set X together which a preorder r such that r up-directs X. We set that up in separate definitions as we sometimes want to use an upward or downward directed set with a partial order rather than a preorder.

definition
```
IsUpDirectedSet(X,r) ≡  IsPreorder(X,r) ∧ (r {up-directs} X)
```

We define the notion of a `downward directed set` analogously.

definition
```
IsDownDirectedSet(X,r) ≡ IsPreorder(X,r) ∧ (r {down-directs} X)
```

We define a linear order as a binary relation that is antisymmetric, transitive and total. Note that this terminology is different than the one used the standard Order.thy file.

definition
```
IsLinOrder(X,r) ≡ antisym(r) ∧ trans(r) ∧ (r {is total on} X)
```

A set is bounded above if there is that is an upper bound for it, i.e. there are some u such that $\langle x, u\rangle \in r$ for all $x \in A$. In addition, the empty set is defined as bounded.

definition
```
IsBoundedAbove(A,r) ≡ ( A=0 ∨ (∃u. ∀x∈A. ⟨x,u⟩ ∈ r))
```

We define sets bounded below analogously.

definition
```
IsBoundedBelow(A,r) ≡ (A=0 ∨ (∃l. ∀x∈A. ⟨l,x⟩ ∈ r))
```

A set is bounded if it is bounded below and above.

definition
```
IsBounded(A,r) ≡ (IsBoundedAbove(A,r) ∧ IsBoundedBelow(A,r))
```

The notation for the definition of an interval may be mysterious for some readers, see lemma `Order_ZF_2_L1` for more intuitive notation.

definition
```
Interval(r,a,b) ≡ r{a} ∩ r-{b}
```

We also define the maximum (the greater of) two elemnts in the obvious way.

definition
```
GreaterOf(r,a,b) ≡ (if ⟨ a,b⟩ ∈ r then b else a)
```

The definition a a minimum (the smaller of) two elements.

definition
```
SmallerOf(r,a,b) ≡ (if ⟨ a,b⟩ ∈ r then a else b)
```

We say that a set has a maximum if it has an element that is not smaller that any other one. We show that under some conditions this element of the set is unique (if exists).

definition
`HasAmaximum(r,A)` $\equiv$ `∃M∈A.∀x∈A. ⟨x,M⟩ ∈ r`

A similar definition what it means that a set has a minimum.

definition
`HasAminimum(r,A)` $\equiv$ `∃m∈A.∀x∈A. ⟨m,x⟩ ∈ r`

Definition of the maximum of a set.

definition
`Maximum(r,A)` $\equiv$ `THE M. M∈A ∧ (∀x∈A. ⟨x,M⟩ ∈ r)`

Definition of a minimum of a set.

definition
`Minimum(r,A)` $\equiv$ `THE m. m∈A ∧ (∀x∈A. ⟨m,x⟩ ∈ r)`

The supremum of a set A is defined as the minimum of the set of upper bounds, i.e. the set $\{u.\forall_{a\in A}\langle a,u\rangle \in r\} = \bigcap_{a\in A} r\{a\}$. Recall that in Isabelle/ZF `r-(A)` denotes the inverse image of the set A by relation r (i.e. `r-(A)`$=\{x : \langle x,y\rangle \in r$ for some $y \in A\}$).

definition
`Supremum(r,A)` $\equiv$ `Minimum(r,⋂a∈A. r{a})`

The notion of "having a supremum" is the same as the set of upper bounds having a minimum, but having it a a separate notion does simplify notation in some cases. The definition is written in terms of images of singletons $\{x\}$ under relation. To understand this formulation note that the set of upper bounds of a set $A \subseteq X$ is $\bigcap_{x\in A}\{y \in X|\langle x,y\rangle \in r\}$, which is the same as $\bigcap_{x\in A} r(\{x\})$, where $r(\{x\})$ is the image of the singleton $\{x\}$ under relation r.

definition
`HasAsupremum(r,A)` $\equiv$ `HasAminimum(r,⋂a∈A. r{a})`

The notion of "having an infimum" is the same as the set of lower bounds having a maximum.

definition
`HasAnInfimum(r,A)` $\equiv$ `HasAmaximum(r,⋂a∈A. r-{a})`

Infimum is defined analogously.

definition
`Infimum(r,A)` $\equiv$ `Maximum(r,⋂a∈A. r-{a})`

We define a relation to be complete if every nonempty bounded above set has a supremum.

definition
`IsComplete (_ {is complete})` **where**
`r {is complete}` $\equiv$

```
∀A. IsBoundedAbove(A,r) ∧ A≠0 ⟶ HasAminimum(r,⋂a∈A. r{a})
```

Normally the "⊆" does not represent a relation, but it does if we restrict it to some colection of sets X. We will call such relation the InclusionOn(X).

```
definition
  InclusionOn(X) ≡ {p∈X×X. fst(p) ⊆ snd(p)}
```

Inclusion relation is a partial order on the powerset of X.

```
lemma incl_is_partorder: shows IsPartOrder(X,InclusionOn(X))
  unfolding InclusionOn_def IsPartOrder_def refl_def antisym_def trans_def
  by auto
```

If a relation down-directs a set, then a larger one does as well.

```
lemma down_dir_mono: assumes r {down-directs} X r⊆R
  shows R {down-directs} X using assms unfolding DownDirects_def
  by blast
```

If a relation up-directs a set, then a larger one does as well.

```
lemma up_dir_mono: assumes r {up-directs} X r⊆R
  shows R {up-directs} X using assms unfolding UpDirects_def
  by blast
```

The essential condition to show that a total relation is reflexive.

```
lemma Order_ZF_1_L1: assumes r {is total on} X and a∈X
  shows ⟨a,a⟩ ∈ r using assms IsTotal_def by auto
```

A total relation is reflexive.

```
lemma total_is_refl:
  assumes r {is total on} X
  shows refl(X,r) using assms Order_ZF_1_L1 refl_def by simp
```

A linear order is partial order.

```
lemma Order_ZF_1_L2: assumes IsLinOrder(X,r)
  shows IsPartOrder(X,r)
  using assms IsLinOrder_def IsPartOrder_def refl_def Order_ZF_1_L1
  by auto
```

Partial order that is total is linear.

```
lemma Order_ZF_1_L3:
  assumes IsPartOrder(X,r) and r {is total on} X
  shows IsLinOrder(X,r)
  using assms IsPartOrder_def IsLinOrder_def
  by simp
```

Relation that is total on a set is total on any subset.

```
lemma Order_ZF_1_L4: assumes r {is total on} X and A⊆X
  shows r {is total on} A
```

```
  using assms IsTotal_def by auto
```

We can restrict a partial order relation to the domain.

```
lemma part_ord_restr: assumes IsPartOrder(X,r)
  shows IsPartOrder(X,r ∩ X×X)
  using assms unfolding IsPartOrder_def refl_def antisym_def trans_def
by auto
```

Partial order on a set implies partial order on a subset.

```
lemma part_ord_subset: assumes IsPartOrder(X,r) and A⊆X
  shows IsPartOrder(A,r)
  using assms unfolding IsPartOrder_def refl_def by auto
```

We can restrict a total order relation to the domain.

```
lemma total_ord_restr: assumes r {is total on} X
  shows (r ∩ X×X) {is total on} X
  using assms unfolding IsTotal_def by auto
```

A linear relation is linear on any subset and we can restrict it to any subset.

```
lemma ord_linear_subset: assumes  IsLinOrder(X,r) and A⊆X
  shows  IsLinOrder(A,r) and  IsLinOrder(A,r ∩ A×A)
proof -
  from assms show IsLinOrder(A,r) using IsLinOrder_def Order_ZF_1_L4
by blast
  then have IsPartOrder(A,r ∩ A×A) and (r ∩ A×A) {is total on} A
    using Order_ZF_1_L2 part_ord_restr total_ord_restr unfolding IsLinOrder_def

    by auto
  then show IsLinOrder(A,r ∩ A×A) using Order_ZF_1_L3 by simp
qed
```

If a relation is a partial order on X and it down-directs a subset of X then that is is a down-directed set.

```
lemma down_directs_subset:
  assumes r {down-directs} A IsPartOrder(X,r) A⊆X
  shows IsDownDirectedSet(A,r)
  using assms part_ord_subset
  unfolding IsPartOrder_def IsPreorder_def IsDownDirectedSet_def
    by simp
```

If the relation is total, then every set is a union of those elements that are nongreater than a given one and nonsmaller than a given one.

```
lemma Order_ZF_1_L5:
  assumes r {is total on} X and A⊆X and a∈X
  shows A = {x∈A. ⟨x,a⟩ ∈ r} ∪ {x∈A. ⟨a,x⟩ ∈ r}
  using assms IsTotal_def by auto
```

A technical fact about reflexive relations.

```
lemma refl_add_point:
  assumes refl(X,r) and A ⊆ B ∪ {x} and B ⊆ X and
  x ∈ X and ∀y∈B. ⟨y,x⟩ ∈ r
  shows ∀a∈A. ⟨a,x⟩ ∈ r
  using assms refl_def by auto
```

5.2 Intervals

In this section we discuss intervals.

The next lemma explains the notation of the definition of an interval.

```
lemma Order_ZF_2_L1:
  shows x ∈ Interval(r,a,b) ⟷ ⟨ a,x⟩ ∈ r ∧ ⟨ x,b⟩ ∈ r
  using Interval_def by auto
```

Since there are some problems with applying the above lemma (seems that simp and auto don't handle equivalence very well), we split `Order_ZF_2_L1` into two lemmas.

```
lemma Order_ZF_2_L1A: assumes x ∈ Interval(r,a,b)
  shows ⟨a,x⟩ ∈ r  ⟨x,b⟩ ∈ r
  using assms  Order_ZF_2_L1 by auto
```

`Order_ZF_2_L1`, implication from right to left.

```
lemma Order_ZF_2_L1B: assumes ⟨a,x⟩ ∈ r  ⟨x,b⟩ ∈ r
  shows x ∈ Interval(r,a,b)
  using assms Order_ZF_2_L1 by simp
```

If the relation is reflexive, the endpoints belong to the interval.

```
lemma Order_ZF_2_L2: assumes refl(X,r)
  and a∈X  b∈X and ⟨a,b⟩ ∈ r
  shows
  a ∈ Interval(r,a,b)
  b ∈ Interval(r,a,b)
  using assms refl_def Order_ZF_2_L1 by auto
```

Under the assumptions of `Order_ZF_2_L2`, the interval is nonempty.

```
lemma Order_ZF_2_L2A: assumes refl(X,r)
  and a∈X  b∈X and ⟨a,b⟩ ∈ r
  shows Interval(r,a,b) ≠ 0
proof -
  from assms have a ∈ Interval(r,a,b)
    using Order_ZF_2_L2 by simp
  then show Interval(r,a,b) ≠ 0 by auto
qed
```

If a, b, c, d are in this order, then $[b, c] \subseteq [a, d]$. We only need trasitivity for this to be true.

```
lemma Order_ZF_2_L3:
  assumes A1: trans(r) and A2:⟨a,b⟩∈r  ⟨b,c⟩∈r  ⟨c,d⟩∈r
shows Interval(r,b,c) ⊆ Interval(r,a,d)
proof
  fix x assume A3: x ∈ Interval(r, b, c)
  note A1
  moreover from A2 A3 have ⟨a,b⟩ ∈ r ∧ ⟨b,x⟩ ∈ r using Order_ZF_2_L1A
    by simp
  ultimately have T1: ⟨a,x⟩ ∈ r by (rule Fol1_L3)
  note A1
  moreover from A2 A3 have ⟨x,c⟩ ∈ r ∧ ⟨c,d⟩ ∈ r using Order_ZF_2_L1A
    by simp
  ultimately have ⟨x,d⟩ ∈ r by (rule Fol1_L3)
  with T1 show x ∈ Interval(r,a,d) using Order_ZF_2_L1B
    by simp
qed
```

For reflexive and antisymmetric relations the interval with equal endpoints consists only of that endpoint.

```
lemma Order_ZF_2_L4:
  assumes A1: refl(X,r) and A2: antisym(r) and A3: a∈X
  shows Interval(r,a,a) = {a}
proof
  from A1 A3 have ⟨ a,a⟩ ∈ r using refl_def by simp
  with A1 A3 show {a} ⊆ Interval(r,a,a) using Order_ZF_2_L2 by simp
  from A2 show Interval(r,a,a) ⊆ {a} using Order_ZF_2_L1A Fol1_L4
    by fast
qed
```

For transitive relations the endpoints have to be in the relation for the interval to be nonempty.

```
lemma Order_ZF_2_L5: assumes A1: trans(r) and A2: ⟨ a,b⟩ ∉ r
  shows Interval(r,a,b) = 0
proof -
  { assume Interval(r,a,b)≠0 then obtain x where x ∈ Interval(r,a,b)
    by auto
  with A1 A2 have False using Order_ZF_2_L1A Fol1_L3 by fast
  } thus thesis by auto
qed
```

If a relation is defined on a set, then intervals are subsets of that set.

```
lemma Order_ZF_2_L6: assumes A1: r ⊆ X×X
  shows Interval(r,a,b) ⊆ X
  using assms Interval_def by auto
```

5.3 Bounded sets

In this section we consider properties of bounded sets.

For reflexive relations singletons are bounded.

```
lemma Order_ZF_3_L1: assumes refl(X,r) and a∈X
  shows IsBounded({a},r)
  using assms refl_def IsBoundedAbove_def IsBoundedBelow_def
    IsBounded_def by auto
```

Sets that are bounded above are contained in the domain of the relation.

```
lemma Order_ZF_3_L1A: assumes r ⊆ X×X
  and IsBoundedAbove(A,r)
  shows A⊆X using assms IsBoundedAbove_def by auto
```

Sets that are bounded below are contained in the domain of the relation.

```
lemma Order_ZF_3_L1B: assumes r ⊆ X×X
  and IsBoundedBelow(A,r)
  shows A⊆X using assms IsBoundedBelow_def by auto
```

For a total relation, the greater of two elements, as defined above, is indeed greater of any of the two.

```
lemma Order_ZF_3_L2: assumes r {is total on} X
  and x∈X y∈X
  shows
  ⟨x,GreaterOf(r,x,y)⟩ ∈ r
  ⟨y,GreaterOf(r,x,y)⟩ ∈ r
  ⟨SmallerOf(r,x,y),x⟩ ∈ r
  ⟨SmallerOf(r,x,y),y⟩ ∈ r
  using assms IsTotal_def Order_ZF_1_L1 GreaterOf_def SmallerOf_def
  by auto
```

If A is bounded above by u, B is bounded above by w, then $A \cup B$ is bounded above by the greater of u, w.

```
lemma Order_ZF_3_L2B:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: u∈X w∈X
  and A4: ∀x∈A. ⟨ x,u⟩ ∈ r ∀x∈B. ⟨ x,w⟩ ∈ r
  shows ∀x∈A∪B. ⟨x,GreaterOf(r,u,w)⟩ ∈ r
proof
  let v = GreaterOf(r,u,w)
  from A1 A3 have T1: ⟨ u,v⟩ ∈ r and T2: ⟨ w,v⟩ ∈ r
    using Order_ZF_3_L2 by auto
  fix x assume A5: x∈A∪B show ⟨x,v⟩ ∈ r
  proof -
    { assume x∈A
    with A4 T1 have ⟨ x,u⟩ ∈ r ∧ ⟨ u,v⟩ ∈ r by simp
    with A2 have ⟨x,v⟩ ∈ r by (rule Fol1_L3) }
  moreover
  { assume x∉A
    with A5 A4 T2 have ⟨ x,w⟩ ∈ r ∧ ⟨ w,v⟩ ∈ r by simp
    with A2 have ⟨x,v⟩ ∈ r by (rule Fol1_L3) }
```

```
  ultimately show thesis by auto
  qed
qed
```

For total and transitive relation the union of two sets bounded above is bounded above.

```
lemma Order_ZF_3_L3:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: IsBoundedAbove(A,r) IsBoundedAbove(B,r)
  and A4: r ⊆ X×X
  shows IsBoundedAbove(A∪B,r)
proof -
  { assume A=0 ∨ B=0
    with A3 have IsBoundedAbove(A∪B,r) by auto }
  moreover
  { assume ¬ (A = 0 ∨ B = 0)
    then have T1: A≠0 B≠0 by auto
    with A3 obtain u w where D1: ∀x∈A. ⟨ x,u⟩ ∈ r ∀x∈B. ⟨ x,w⟩ ∈ r
      using IsBoundedAbove_def by auto
    let U = GreaterOf(r,u,w)
    from T1 A4 D1 have u∈X w∈X by auto
    with A1 A2 D1 have ∀x∈A∪B.⟨ x,U⟩ ∈ r
      using Order_ZF_3_L2B by blast
    then have IsBoundedAbove(A∪B,r)
      using IsBoundedAbove_def by auto }
  ultimately show thesis by auto
qed
```

For total and transitive relations if a set A is bounded above then $A \cup \{a\}$ is bounded above.

```
lemma Order_ZF_3_L4:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: IsBoundedAbove(A,r) and A4: a∈X and A5: r ⊆ X×X
  shows IsBoundedAbove(A∪{a},r)
proof -
  from A1 have refl(X,r)
    using total_is_refl by simp
  with assms show thesis using
    Order_ZF_3_L1 IsBounded_def Order_ZF_3_L3 by simp
qed
```

If A is bounded below by l, B is bounded below by m, then $A \cup B$ is bounded below by the smaller of u, w.

```
lemma Order_ZF_3_L5B:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: l∈X m∈X
  and A4: ∀x∈A. ⟨ l,x⟩ ∈ r ∀x∈B. ⟨ m,x⟩ ∈ r
  shows ∀x∈A∪B. ⟨SmallerOf(r,l,m),x⟩ ∈ r
```

```
proof
  let k = SmallerOf(r,l,m)
  from A1 A3 have T1: ⟨ k,l⟩ ∈ r and T2: ⟨ k,m⟩ ∈ r
    using Order_ZF_3_L2 by auto
  fix x assume A5: x∈A∪B show ⟨k,x⟩ ∈ r
  proof -
    { assume x∈A
      with A4 T1 have ⟨ k,l⟩ ∈ r ∧ ⟨ l,x⟩ ∈ r by simp
      with A2 have ⟨k,x⟩ ∈ r by (rule Fol1_L3) }
    moreover
    { assume x∉A
      with A5 A4 T2 have ⟨ k,m⟩ ∈ r ∧ ⟨ m,x⟩ ∈ r by simp
      with A2 have ⟨k,x⟩ ∈ r by (rule Fol1_L3) }
    ultimately show thesis by auto
  qed
qed
```

For total and transitive relation the union of two sets bounded below is bounded below.

```
lemma Order_ZF_3_L6:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: IsBoundedBelow(A,r) IsBoundedBelow(B,r)
  and A4: r ⊆ X×X
  shows IsBoundedBelow(A∪B,r)
proof -
  { assume A=0 ∨ B=0
    with A3 have thesis by auto }
  moreover
  { assume ¬ (A = 0 ∨ B = 0)
    then have T1: A≠0 B≠0 by auto
    with A3 obtain l m where D1: ∀x∈A. ⟨ l,x⟩ ∈ r ∀x∈B. ⟨ m,x⟩ ∈ r
      using IsBoundedBelow_def by auto
    let L = SmallerOf(r,l,m)
    from T1 A4 D1 have T1: l∈X m∈X by auto
    with A1 A2 D1 have ∀x∈A∪B.⟨ L,x⟩ ∈ r
      using Order_ZF_3_L5B by blast
    then have IsBoundedBelow(A∪B,r)
      using IsBoundedBelow_def by auto }
  ultimately show thesis by auto
qed
```

For total and transitive relations if a set A is bounded below then $A \cup \{a\}$ is bounded below.

```
lemma Order_ZF_3_L7:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: IsBoundedBelow(A,r) and A4: a∈X and A5: r ⊆ X×X
  shows IsBoundedBelow(A∪{a},r)
proof -
  from A1 have refl(X,r)
```

```
    using total_is_refl by simp
  with assms show thesis using
    Order_ZF_3_L1 IsBounded_def Order_ZF_3_L6 by simp
qed
```

For total and transitive relations unions of two bounded sets are bounded.

```
theorem Order_ZF_3_T1:
  assumes r {is total on} X and trans(r)
  and IsBounded(A,r) IsBounded(B,r)
  and r ⊆ X×X
  shows IsBounded(A∪B,r)
  using assms Order_ZF_3_L3 Order_ZF_3_L6 Order_ZF_3_L7 IsBounded_def
  by simp
```

For total and transitive relations if a set A is bounded then $A \cup \{a\}$ is bounded.

```
lemma Order_ZF_3_L8:
  assumes r {is total on} X and trans(r)
  and IsBounded(A,r) and a∈X and r ⊆ X×X
  shows IsBounded(A∪{a},r)
  using assms total_is_refl Order_ZF_3_L1 Order_ZF_3_T1 by blast
```

A sufficient condition for a set to be bounded below.

```
lemma Order_ZF_3_L9: assumes A1: ∀a∈A. ⟨l,a⟩ ∈ r
  shows IsBoundedBelow(A,r)
proof -
  from A1 have ∃l. ∀x∈A. ⟨l,x⟩ ∈ r
    by auto
  then show IsBoundedBelow(A,r)
    using IsBoundedBelow_def by simp
qed
```

A sufficient condition for a set to be bounded above.

```
lemma Order_ZF_3_L10: assumes A1: ∀a∈A. ⟨a,u⟩ ∈ r
  shows IsBoundedAbove(A,r)
proof -
  from A1 have ∃u. ∀x∈A. ⟨x,u⟩ ∈ r
    by auto
  then show IsBoundedAbove(A,r)
    using IsBoundedAbove_def by simp
qed
```

Intervals are bounded.

```
lemma Order_ZF_3_L11: shows
  IsBoundedAbove(Interval(r,a,b),r)
  IsBoundedBelow(Interval(r,a,b),r)
  IsBounded(Interval(r,a,b),r)
proof -
```

```
  { fix x assume x ∈ Interval(r,a,b)
    then have ⟨ x,b⟩ ∈ r  ⟨ a,x⟩ ∈ r
      using Order_ZF_2_L1A by auto
  } then have
      ∃u. ∀x∈Interval(r,a,b). ⟨ x,u⟩ ∈ r
      ∃l. ∀x∈Interval(r,a,b). ⟨ l,x⟩ ∈ r
    by auto
  then show
    IsBoundedAbove(Interval(r,a,b),r)
    IsBoundedBelow(Interval(r,a,b),r)
    IsBounded(Interval(r,a,b),r)
    using IsBoundedAbove_def IsBoundedBelow_def IsBounded_def
    by auto
qed
```

A subset of a set that is bounded below is bounded below.

```
lemma Order_ZF_3_L12: assumes A1: IsBoundedBelow(A,r) and A2: B⊆A
  shows IsBoundedBelow(B,r)
proof -
  { assume A = 0
    with assms have IsBoundedBelow(B,r)
      using IsBoundedBelow_def by auto }
  moreover
  { assume A ≠ 0
    with A1 have ∃l. ∀x∈A. ⟨l,x⟩ ∈ r
      using IsBoundedBelow_def by simp
    with A2 have ∃l.∀x∈B. ⟨l,x⟩ ∈ r by auto
    then have IsBoundedBelow(B,r) using IsBoundedBelow_def
      by auto }
  ultimately show IsBoundedBelow(B,r) by auto
qed
```

A subset of a set that is bounded above is bounded above.

```
lemma Order_ZF_3_L13: assumes A1: IsBoundedAbove(A,r) and A2: B⊆A
  shows IsBoundedAbove(B,r)
proof -
  { assume A = 0
    with assms have IsBoundedAbove(B,r)
      using IsBoundedAbove_def by auto }
  moreover
  { assume A ≠ 0
    with A1 have ∃u. ∀x∈A. ⟨x,u⟩ ∈ r
      using IsBoundedAbove_def by simp
    with A2 have ∃u.∀x∈B. ⟨x,u⟩ ∈ r by auto
    then have IsBoundedAbove(B,r) using IsBoundedAbove_def
      by auto }
  ultimately show IsBoundedAbove(B,r) by auto
qed
```

If for every element of X we can find one in A that is greater, then the A

can not be bounded above. Works for relations that are total, transitive and antisymmetric, (i.e. for linear order relations).

```
lemma Order_ZF_3_L14:
  assumes A1: r {is total on} X
  and A2: trans(r) and A3: antisym(r)
  and A4: r ⊆ X×X and A5: X≠0
  and A6: ∀x∈X. ∃a∈A. x≠a ∧ ⟨x,a⟩ ∈ r
  shows ¬IsBoundedAbove(A,r)
proof -
  { from A5 A6 have I: A≠0 by auto
    moreover assume IsBoundedAbove(A,r)
    ultimately obtain u where II: ∀x∈A. ⟨ x,u⟩ ∈ r
      using IsBounded_def IsBoundedAbove_def by auto
    with A4 I have u∈X by auto
    with A6 obtain b where b∈A and III: u≠b and ⟨u,b⟩ ∈ r
      by auto
    with II have ⟨b,u⟩ ∈ r  ⟨u,b⟩ ∈ r by auto
    with A3 have b=u by (rule Fol1_L4)
    with III have False by simp
  } thus ¬IsBoundedAbove(A,r) by auto
qed
```

The set of elements in a set A that are nongreater than a given element is bounded above.

```
lemma Order_ZF_3_L15: shows IsBoundedAbove({x∈A. ⟨x,a⟩ ∈ r},r)
  using IsBoundedAbove_def by auto
```

If A is bounded below, then the set of elements in a set A that are nongreater than a given element is bounded.

```
lemma Order_ZF_3_L16: assumes A1: IsBoundedBelow(A,r)
  shows IsBounded({x∈A. ⟨x,a⟩ ∈ r},r)
proof -
  { assume A=0
    then have IsBounded({x∈A. ⟨x,a⟩ ∈ r},r)
      using IsBoundedBelow_def IsBoundedAbove_def IsBounded_def
      by auto }
  moreover
  { assume A≠0
    with A1 obtain l where I: ∀x∈A. ⟨l,x⟩ ∈ r
      using IsBoundedBelow_def by auto
    then have ∀y∈{x∈A. ⟨x,a⟩ ∈ r}. ⟨l,y⟩ ∈ r by simp
    then have IsBoundedBelow({x∈A. ⟨x,a⟩ ∈ r},r)
      by (rule Order_ZF_3_L9)
    then have IsBounded({x∈A. ⟨x,a⟩ ∈ r},r)
      using Order_ZF_3_L15 IsBounded_def by simp }
  ultimately show thesis by blast
qed

end
```

6 More on order relations

theory `Order_ZF_1` **imports** `ZF.Order ZF1`

begin

In `Order_ZF` we define some notions related to order relations based on the nonstrict orders (≤ type). Some people however prefer to talk about these notions in terms of the strict order relation (< type). This is the case for the standard Isabelle `Order.thy` and also for Metamath. In this theory file we repeat some developments from `Order_ZF` using the strict order relation as a basis.This is mostly useful for Metamath translation, but is also of some general interest. The names of theorems are copied from Metamath.

6.1 Definitions and basic properties

In this section we introduce some definitions taken from Metamath and relate them to the ones used by the standard Isabelle `Order.thy`.

The next definition is the strict version of the linear order. What we write as `R Orders A` is written $ROrdA$ in Metamath.

```
definition
StrictOrder (infix Orders 65) where
  R Orders A ≡ ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
  (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
  (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
```

The definition of supremum for a (strict) linear order.

```
definition
  Sup(B,A,R) ≡
  ⋃ {x ∈ A. (∀y∈B. ⟨x,y⟩ ∉ R) ∧
  (∀y∈A. ⟨y,x⟩ ∈ R ⟶ (∃z∈B. ⟨y,z⟩ ∈ R))}
```

Definition of infimum for a linear order. It is defined in terms of supremum.

```
definition
  Infim(B,A,R) ≡ Sup(B,A,converse(R))
```

If relation R orders a set A, (in Metamath sense) then R is irreflexive, transitive and linear therefore is a total order on A (in Isabelle sense).

```
lemma orders_imp_tot_ord: assumes A1: R Orders A
  shows
  irrefl(A,R)
  trans[A](R)
  part_ord(A,R)
  linear(A,R)
  tot_ord(A,R)
proof -
```

```
  from A1 have I:
    ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
    (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
    (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
    unfolding StrictOrder_def by simp
  then have ∀x∈A. ⟨x,x⟩ ∉ R by blast
  then show irrefl(A,R) using irrefl_def by simp
  moreover
  from I have
    ∀x∈A. ∀y∈A. ∀z∈A. ⟨x,y⟩ ∈ R ⟶ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R
    by blast
  then show trans[A](R) unfolding trans_on_def by blast
  ultimately show part_ord(A,R) using part_ord_def
    by simp
  moreover
  from I have
    ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ R ∨ x=y ∨ ⟨y,x⟩ ∈ R
    by blast
  then show linear(A,R) unfolding linear_def by blast
  ultimately show tot_ord(A,R) using tot_ord_def
    by simp
qed
```

A converse of `orders_imp_tot_ord`. Together with that theorem this shows that Metamath's notion of an order relation is equivalent to Isabelles `tot_ord` predicate.

```
lemma tot_ord_imp_orders: assumes A1: tot_ord(A,R)
  shows R Orders A
proof -
  from A1 have
    I: linear(A,R) and
    II: irrefl(A,R) and
    III: trans[A](R) and
    IV: part_ord(A,R)
    using tot_ord_def part_ord_def by auto
  from IV have asym(R ∩ A×A)
    using part_ord_Imp_asym by simp
  then have V: ∀x y. ⟨x,y⟩ ∈ (R ∩ A×A) ⟶ ¬(⟨y,x⟩ ∈ (R ∩ A×A))
    unfolding asym_def by blast
  from I have VI: ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ R ∨ x=y ∨ ⟨y,x⟩ ∈ R
    unfolding linear_def by blast
  from III have VII:
    ∀x∈A. ∀y∈A. ∀z∈A. ⟨x,y⟩ ∈ R ⟶ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R
    unfolding trans_on_def by blast
  { fix x y z
    assume T: x∈A y∈A z∈A
    have ⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)
    proof
      assume A2: ⟨x,y⟩ ∈ R
```

```
      with V T have  ¬(⟨y,x⟩ ∈ R) by blast
      moreover from II T A2 have x≠y using irrefl_def
 by auto
      ultimately show ¬(x=y ∨ ⟨y,x⟩ ∈ R) by simp
    next assume  ¬(x=y ∨ ⟨y,x⟩ ∈ R)
      with VI T show ⟨x,y⟩ ∈ R by auto
    qed
    moreover from VII T have
      ⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R
      by blast
    ultimately have (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
      (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
      by simp
  } then have ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
      (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
      (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
    by auto
  then show R Orders A using StrictOrder_def by simp
qed
```

6.2 Properties of (strict) total orders

In this section we discuss the properties of strict order relations. This continues the development contained in the standard Isabelle's `Order.thy` with a view towards using the theorems translated from Metamath.

A relation orders a set iff the converse relation orders a set. Going one way we can use the the lemma `tot_od_converse` from the standard Isabelle's `Order.thy`.The other way is a bit more complicated (note that in Isabelle for `converse(converse(r)) = r` one needs r to consist of ordered pairs, which does not follow from the `StrictOrder` definition above).

```
lemma cnvso: shows R Orders A ⟷ converse(R) Orders A
proof
  let r = converse(R)
  assume R Orders A
  then have tot_ord(A,r) using orders_imp_tot_ord tot_ord_converse
    by simp
  then show r Orders A using tot_ord_imp_orders
    by simp
next
  let r = converse(R)
  assume r Orders A
  then have A2: ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
    (⟨x,y⟩ ∈ r ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ r)) ∧
    (⟨x,y⟩ ∈ r ∧ ⟨y,z⟩ ∈ r ⟶ ⟨x,z⟩ ∈ r)
    using StrictOrder_def by simp
  { fix x y z
    assume x∈A ∧ y∈A ∧ z∈A
```

```
    with A2 have
      I: ⟨y,x⟩ ∈ r ⟷ ¬(x=y ∨ ⟨x,y⟩ ∈ r) and
      II: ⟨y,x⟩ ∈ r ∧ ⟨z,y⟩ ∈ r ⟶ ⟨z,x⟩ ∈ r
      by auto
    from I have ⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)
      by auto
    moreover from II have ⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R
      by auto
    ultimately have (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
      (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R) by simp
  } then have  ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
      (⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
      (⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
    by auto
  then show R Orders A using StrictOrder_def by simp
qed
```

Supremum is unique, if it exists.

```
lemma supeu: assumes A1: R Orders A and A2: x∈A and
  A3: ∀y∈B. ⟨x,y⟩ ∉ R and A4: ∀y∈A. ⟨y,x⟩ ∈ R  ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R)
   shows
  ∃!x. x∈A∧(∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈
R))
proof
  from A2 A3 A4 show
    ∃ x. x∈A∧(∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩
∈ R))
    by auto
next fix x1 x2
  assume A5:
    x1 ∈ A ∧ (∀y∈B. ⟨x1,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x1⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩
∈ R))
    x2 ∈ A ∧ (∀y∈B. ⟨x2,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x2⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩
∈ R))
  from A1 have linear(A,R) using orders_imp_tot_ord tot_ord_def
    by simp
  then have ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ R ∨ x=y ∨ ⟨y,x⟩ ∈ R
    unfolding linear_def by blast
  with A5 have ⟨x1,x2⟩ ∈ R ∨ x1=x2 ∨ ⟨x2,x1⟩ ∈ R by blast
  moreover
  { assume ⟨x1,x2⟩ ∈ R
    with A5 obtain z where z∈B and ⟨x1,z⟩ ∈ R by auto
    with A5 have False by auto }
  moreover
  { assume ⟨x2,x1⟩ ∈ R
    with A5 obtain z where z∈B and ⟨x2,z⟩ ∈ R by auto
    with A5 have False by auto }
  ultimately  show x1 = x2 by auto
qed
```

Supremum has expected properties if it exists.

```
lemma sup_props: assumes A1: R Orders A and
  A2: ∃x∈A. (∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈
R))
  shows
  Sup(B,A,R) ∈ A
  ∀y∈B. ⟨Sup(B,A,R),y⟩ ∉ R
  ∀y∈A. ⟨y,Sup(B,A,R)⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R )
proof -
  let S = {x∈A. (∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩
∈ R ) ) }
  from A2 obtain x where
    x∈A and (∀y∈B. ⟨x,y⟩ ∉ R) and ∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩
∈ R)
    by auto
  with A1 have I:
    ∃!x. x∈A∧(∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩
∈ R))
    using supeu by simp
  then have ( ⋃S ) ∈ A by (rule ZF1_1_L9)
  then show Sup(B,A,R) ∈ A using Sup_def by simp
  from I have II:
    (∀y∈B. ⟨⋃S ,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,⋃S⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R))
    by (rule ZF1_1_L9)
  hence ∀y∈B. ⟨⋃S,y⟩ ∉ R by blast
  moreover have III: (⋃S) = Sup(B,A,R) using Sup_def by simp
  ultimately show ∀y∈B. ⟨Sup(B,A,R),y⟩ ∉ R by simp
  from II have IV: ∀y∈A. ⟨y,⋃S⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R)
    by blast
  { fix y assume A3: y∈A and ⟨y,Sup(B,A,R)⟩ ∈ R
    with III have ⟨y,⋃S⟩ ∈ R by simp
    with IV A3 have ∃z∈B. ⟨y,z⟩ ∈ R by blast
  } thus  ∀y∈A. ⟨y,Sup(B,A,R)⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R )
    by simp
qed
```

Elements greater or equal than any element of B are greater or equal than supremum of B.

```
lemma supnub: assumes A1: R Orders A and A2:
  ∃x∈A. (∀y∈B. ⟨x,y⟩ ∉ R) ∧ (∀y∈A. ⟨y,x⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R))
  and A3: c ∈ A and A4: ∀z∈B. ⟨c,z⟩ ∉ R
  shows ⟨c, Sup(B,A,R)⟩ ∉ R
proof -
  from A1 A2 have
    ∀y∈A. ⟨y,Sup(B,A,R)⟩ ∈ R ⟶ ( ∃z∈B. ⟨y,z⟩ ∈ R )
    by (rule sup_props)
  with A3 A4 show ⟨c, Sup(B,A,R)⟩ ∉ R by auto
qed
```

end

7 Even more on order relations

theory Order_ZF_1a imports Order_ZF

begin

This theory is a continuation of Order_ZF and talks about maximum and minimum of a set, supremum and infimum and strict (not reflexive) versions of order relations.

7.1 Maximum and minimum of a set

In this section we show that maximum and minimum are unique if they exist. We also show that union of sets that have maxima (minima) has a maximum (minimum). We also show that singletons have maximum and minimum. All this allows to show (in Finite_ZF) that every finite set has well-defined maximum and minimum.

A somewhat technical fact that allows to reduce the number of premises in some theorems: the assumption that a set has a maximum implies that it is not empty.

```
lemma set_max_not_empty: assumes HasAmaximum(r,A) shows A≠0
  using assms unfolding HasAmaximum_def by auto
```

If a set has a maximum implies that it is not empty.

```
lemma set_min_not_empty: assumes HasAminimum(r,A) shows A≠0
  using assms unfolding HasAminimum_def by auto
```

If a set has a supremum then it cannot be empty. We are probably using the fact that $\bigcap \emptyset = \emptyset$, which makes me a bit anxious as this I think is just a convention.

```
lemma set_sup_not_empty: assumes HasAsupremum(r,A) shows A≠0
proof -
  from assms have HasAminimum(r,⋂a∈A. r{a}) unfolding HasAsupremum_def
    by simp
  then have (⋂a∈A. r{a}) ≠ 0 using set_min_not_empty by simp
  then obtain x where x ∈ (⋂y∈A. r{y}) by blast
  thus thesis by auto
qed
```

If a set has an infimum then it cannot be empty.

```
lemma set_inf_not_empty: assumes HasAnInfimum(r,A) shows A≠0
proof -
  from assms have HasAmaximum(r,⋂a∈A. r-{a}) unfolding HasAnInfimum_def
```

```
    by simp
  then have (⋂a∈A. r-{a}) ≠ 0 using set_max_not_empty by simp
  then obtain x where x ∈ (⋂y∈A. r-{y}) by blast
  thus thesis by auto
qed
```

For antisymmetric relations maximum of a set is unique if it exists.

```
lemma Order_ZF_4_L1: assumes A1: antisym(r) and A2: HasAmaximum(r,A)
  shows ∃!M. M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r)
proof
  from A2 show ∃M. M ∈ A ∧ (∀x∈A. ⟨x, M⟩ ∈ r)
    using HasAmaximum_def by auto
  fix M1 M2 assume
    A2: M1 ∈ A ∧ (∀x∈A. ⟨x, M1⟩ ∈ r) M2 ∈ A ∧ (∀x∈A. ⟨x, M2⟩ ∈ r)
    then have ⟨M1,M2⟩ ∈ r ⟨M2,M1⟩ ∈ r by auto
    with A1 show M1=M2 by (rule Fol1_L4)
qed
```

For antisymmetric relations minimum of a set is unique if it exists.

```
lemma Order_ZF_4_L2: assumes A1: antisym(r) and A2: HasAminimum(r,A)
  shows ∃!m. m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r)
proof
  from A2 show ∃m. m ∈ A ∧ (∀x∈A. ⟨m, x⟩ ∈ r)
    using HasAminimum_def by auto
  fix m1 m2 assume
    A2: m1 ∈ A ∧ (∀x∈A. ⟨m1, x⟩ ∈ r) m2 ∈ A ∧ (∀x∈A. ⟨m2, x⟩ ∈ r)
    then have ⟨m1,m2⟩ ∈ r ⟨m2,m1⟩ ∈ r by auto
    with A1 show m1=m2 by (rule Fol1_L4)
qed
```

Maximum of a set has desired properties.

```
lemma Order_ZF_4_L3: assumes A1: antisym(r) and A2: HasAmaximum(r,A)
  shows Maximum(r,A) ∈ A ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
proof -
  let Max = THE M. M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r)
  from A1 A2 have ∃!M. M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r)
    by (rule Order_ZF_4_L1)
  then have Max ∈ A ∧ (∀x∈A. ⟨ x,Max⟩ ∈ r)
    by (rule theI)
  then show Maximum(r,A) ∈ A ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
    using Maximum_def by auto
qed
```

Minimum of a set has desired properties.

```
lemma Order_ZF_4_L4: assumes A1: antisym(r) and A2: HasAminimum(r,A)
  shows Minimum(r,A) ∈ A ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
proof -
  let Min = THE m. m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r)
```

```
  from A1 A2 have ∃!m. m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r)
    by (rule Order_ZF_4_L2)
  then have Min ∈ A ∧ (∀x∈A. ⟨ Min,x⟩ ∈ r)
    by (rule theI)
  then show Minimum(r,A) ∈ A ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
    using Minimum_def by auto
qed
```

For total and transitive relations a union a of two sets that have maxima has a maximum.

```
lemma Order_ZF_4_L5:
  assumes A1: r {is total on} (A∪B) and A2: trans(r)
  and A3: HasAmaximum(r,A) HasAmaximum(r,B)
  shows HasAmaximum(r,A∪B)
proof -
  from A3 obtain M K where
    D1: M∈A ∧ (∀x∈A. ⟨ x,M⟩ ∈ r) K∈B ∧ (∀x∈B. ⟨ x,K⟩ ∈ r)
    using HasAmaximum_def by auto
  let L = GreaterOf(r,M,K)
  from D1 have T1: M ∈ A∪B K ∈ A∪B
    ∀x∈A. ⟨ x,M⟩ ∈ r ∀x∈B. ⟨ x,K⟩ ∈ r
    by auto
  with A1 A2 have ∀x∈A∪B.⟨ x,L⟩ ∈ r by (rule Order_ZF_3_L2B)
  moreover from T1 have L ∈ A∪B using GreaterOf_def IsTotal_def
    by simp
  ultimately show HasAmaximum(r,A∪B) using HasAmaximum_def by auto
qed
```

For total and transitive relations A union a of two sets that have minima has a minimum.

```
lemma Order_ZF_4_L6:
  assumes A1: r {is total on} (A∪B) and A2: trans(r)
  and A3: HasAminimum(r,A) HasAminimum(r,B)
  shows HasAminimum(r,A∪B)
proof -
  from A3 obtain m k where
    D1: m∈A ∧ (∀x∈A. ⟨ m,x⟩ ∈ r) k∈B ∧ (∀x∈B. ⟨ k,x⟩ ∈ r)
    using HasAminimum_def by auto
  let l = SmallerOf(r,m,k)
  from D1 have T1: m ∈ A∪B k ∈ A∪B
    ∀x∈A. ⟨ m,x⟩ ∈ r ∀x∈B. ⟨ k,x⟩ ∈ r
    by auto
  with A1 A2 have ∀x∈A∪B.⟨ l,x⟩ ∈ r by (rule Order_ZF_3_L5B)
  moreover from T1 have l ∈ A∪B using SmallerOf_def IsTotal_def
    by simp
  ultimately show HasAminimum(r,A∪B) using HasAminimum_def by auto
qed
```

Set that has a maximum is bounded above.

```
lemma Order_ZF_4_L7:
  assumes HasAmaximum(r,A)
  shows IsBoundedAbove(A,r)
  using assms HasAmaximum_def IsBoundedAbove_def by auto
```

Set that has a minimum is bounded below.

```
lemma Order_ZF_4_L8A:
  assumes HasAminimum(r,A)
  shows IsBoundedBelow(A,r)
  using assms HasAminimum_def IsBoundedBelow_def by auto
```

A subset of a set that has a maximum is bounded above.

```
lemma max_subset_bounded: assumes HasAmaximum(r,A) and B⊆A
  shows IsBoundedAbove(B,r)
proof -
  from assms(1) obtain M where ∀x∈A. ⟨x,M⟩ ∈ r
    unfolding HasAmaximum_def by auto
  with assms(2) show thesis unfolding IsBoundedAbove_def by blast
qed
```

A subset of a set that has a minimum is bounded below.

```
lemma min_subset_bounded: assumes HasAminimum(r,A) and B⊆A
  shows IsBoundedBelow(B,r)
proof -
  from assms(1) obtain m where ∀x∈A. ⟨m,x⟩ ∈ r
    unfolding HasAminimum_def by auto
  with assms(2) show thesis unfolding IsBoundedBelow_def by blast
qed
```

For reflexive relations singletons have a minimum and maximum.

```
lemma Order_ZF_4_L8: assumes refl(X,r) and a∈X
  shows HasAmaximum(r,{a}) HasAminimum(r,{a})
  using assms refl_def HasAmaximum_def HasAminimum_def by auto
```

For total and transitive relations if we add an element to a set that has a maximum, the set still has a maximum.

```
lemma Order_ZF_4_L9:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: A⊆X and A4: a∈X and A5: HasAmaximum(r,A)
  shows HasAmaximum(r,A∪{a})
proof -
  from A3 A4 have A∪{a} ⊆ X by auto
  with A1 have r {is total on} (A∪{a})
    using Order_ZF_1_L4 by blast
  moreover from A1 A2 A4 A5 have
    trans(r) HasAmaximum(r,A) by auto
  moreover from A1 A4 have HasAmaximum(r,{a})
    using total_is_refl Order_ZF_4_L8 by blast
```

```
  ultimately show HasAmaximum(r,A∪{a}) by (rule Order_ZF_4_L5)
qed
```

For total and transitive relations if we add an element to a set that has a minimum, the set still has a minimum.

```
lemma Order_ZF_4_L10:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: A⊆X and A4: a∈X and A5: HasAminimum(r,A)
  shows HasAminimum(r,A∪{a})
proof -
  from A3 A4 have A∪{a} ⊆ X by auto
  with A1 have r {is total on} (A∪{a})
    using Order_ZF_1_L4 by blast
  moreover from A1 A2 A4 A5 have
    trans(r) HasAminimum(r,A) by auto
  moreover from A1 A4 have HasAminimum(r,{a})
    using total_is_refl Order_ZF_4_L8 by blast
  ultimately show HasAminimum(r,A∪{a}) by (rule Order_ZF_4_L6)
qed
```

If the order relation has a property that every nonempty bounded set attains a minimum (for example integers are like that), then every nonempty set bounded below attains a minimum.

```
lemma Order_ZF_4_L11:
  assumes A1: r {is total on} X and
  A2: trans(r) and
  A3: r ⊆ X×X and
  A4: ∀A. IsBounded(A,r) ∧ A≠0 ⟶ HasAminimum(r,A) and
  A5: B≠0 and A6: IsBoundedBelow(B,r)
  shows HasAminimum(r,B)
proof -
  from A5 obtain b where T: b∈B by auto
  let L = {x∈B. ⟨x,b⟩ ∈ r}
  from A3 A6 T have T1: b∈X using Order_ZF_3_L1B by blast
  with A1 T have T2: b ∈ L
    using total_is_refl refl_def by simp
  then have L ≠ 0 by auto
  moreover have IsBounded(L,r)
  proof -
    have L ⊆ B by auto
    with A6 have IsBoundedBelow(L,r)
      using Order_ZF_3_L12 by simp
    moreover have IsBoundedAbove(L,r)
      by (rule Order_ZF_3_L15)
    ultimately have IsBoundedAbove(L,r) ∧ IsBoundedBelow(L,r)
      by blast
    then show IsBounded(L,r) using IsBounded_def
      by simp
  qed
```

```
  ultimately have IsBounded(L,r) ∧ L ≠ 0 by blast
  with A4 have HasAminimum(r,L) by simp
  then obtain m where I: m∈L and II: ∀x∈L. ⟨ m,x⟩ ∈ r
    using HasAminimum_def by auto
  then have III: ⟨m,b⟩ ∈ r by simp
  from I have m∈B by simp
  moreover have ∀x∈B. ⟨m,x⟩ ∈ r
  proof
    fix x assume A7: x∈B
    from A3 A6 have B⊆X using Order_ZF_3_L1B by blast
    with A1 A7 T1 have x ∈  L ∪ {x∈B. ⟨b,x⟩ ∈ r}
      using Order_ZF_1_L5 by simp
    then have x∈L ∨ ⟨b,x⟩ ∈ r by auto
    moreover
    { assume x∈L
      with II have ⟨m,x⟩ ∈ r by simp }
    moreover
    { assume ⟨b,x⟩ ∈ r
      with A2 III have trans(r) and ⟨m,b⟩ ∈ r ∧ ⟨b,x⟩ ∈ r
 by auto
      then have  ⟨m,x⟩ ∈ r by (rule Fol1_L3) }
    ultimately show ⟨m,x⟩ ∈ r by auto
  qed
  ultimately show HasAminimum(r,B) using HasAminimum_def
    by auto
qed
```

A dual to Order_ZF_4_L11: If the order relation has a property that every nonempty bounded set attains a maximum (for example integers are like that), then every nonempty set bounded above attains a maximum.

```
lemma Order_ZF_4_L11A:
  assumes A1: r {is total on} X and
  A2: trans(r) and
  A3: r ⊆ X×X and
  A4: ∀A. IsBounded(A,r) ∧ A≠0 ⟶ HasAmaximum(r,A) and
  A5: B≠0 and A6: IsBoundedAbove(B,r)
  shows HasAmaximum(r,B)
proof -
  from A5 obtain b where T: b∈B by auto
  let U = {x∈B. ⟨b,x⟩ ∈ r}
  from A3 A6 T have T1: b∈X using Order_ZF_3_L1A by blast
  with A1 T have T2: b ∈ U
    using total_is_refl refl_def by simp
  then have U ≠ 0 by auto
  moreover have IsBounded(U,r)
  proof -
    have U ⊆ B by auto
    with A6 have IsBoundedAbove(U,r)
      using Order_ZF_3_L13 by blast
```

```
    moreover have IsBoundedBelow(U,r)
      using IsBoundedBelow_def by auto
    ultimately have IsBoundedAbove(U,r) ∧ IsBoundedBelow(U,r)
      by blast
    then show IsBounded(U,r) using IsBounded_def
      by simp
  qed
  ultimately have IsBounded(U,r) ∧ U ≠ 0 by blast
  with A4 have HasAmaximum(r,U) by simp
  then obtain m where I: m∈U and II: ∀x∈U. ⟨x,m⟩ ∈ r
    using HasAmaximum_def by auto
  then have III: ⟨b,m⟩ ∈ r by simp
  from I have m∈B by simp
  moreover have ∀x∈B. ⟨x,m⟩ ∈ r
  proof
    fix x assume A7: x∈B
    from A3 A6 have B⊆X using Order_ZF_3_L1A by blast
    with A1 A7 T1 have x ∈ {x∈B. ⟨x,b⟩ ∈ r} ∪ U
      using Order_ZF_1_L5 by simp
    then have x∈U ∨ ⟨x,b⟩ ∈ r by auto
    moreover
    { assume x∈U
      with II have ⟨x,m⟩ ∈ r by simp }
    moreover
    { assume ⟨x,b⟩ ∈ r
      with A2 III have trans(r) and ⟨x,b⟩ ∈ r ∧ ⟨b,m⟩ ∈ r
 by auto
      then have  ⟨x,m⟩ ∈ r by (rule Fol1_L3) }
    ultimately show ⟨x,m⟩ ∈ r by auto
  qed
  ultimately show HasAmaximum(r,B) using HasAmaximum_def
    by auto
qed
```

If a set has a minimum and L is less or equal than all elements of the set, then L is less or equal than the minimum.

```
lemma Order_ZF_4_L12:
  assumes antisym(r) and HasAminimum(r,A) and ∀a∈A. ⟨L,a⟩ ∈ r
  shows ⟨L,Minimum(r,A)⟩ ∈ r
  using assms Order_ZF_4_L4 by simp
```

If a set has a maximum and all its elements are less or equal than M, then the maximum of the set is less or equal than M.

```
lemma Order_ZF_4_L13:
  assumes antisym(r) and HasAmaximum(r,A) and ∀a∈A. ⟨a,M⟩ ∈ r
  shows ⟨Maximum(r,A),M⟩ ∈ r
  using assms Order_ZF_4_L3 by simp
```

If an element belongs to a set and is greater or equal than all elements of that set, then it is the maximum of that set.

```
lemma Order_ZF_4_L14:
  assumes A1: antisym(r) and A2: M ∈ A and
  A3: ∀a∈A. ⟨a,M⟩ ∈ r
  shows Maximum(r,A) = M
proof -
  from A2 A3 have I: HasAmaximum(r,A) using HasAmaximum_def
    by auto
  with A1 have ∃!M. M∈A ∧ (∀x∈A. ⟨x,M⟩ ∈ r)
    using Order_ZF_4_L1 by simp
  moreover from A2 A3 have M∈A ∧ (∀x∈A. ⟨x,M⟩ ∈ r) by simp
  moreover from A1 I have
    Maximum(r,A) ∈ A ∧ (∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r)
    using Order_ZF_4_L3 by simp
  ultimately show Maximum(r,A) = M by auto
qed
```

If an element belongs to a set and is less or equal than all elements of that set, then it is the minimum of that set.

```
lemma Order_ZF_4_L15:
  assumes A1: antisym(r) and A2: m ∈ A and
  A3: ∀a∈A. ⟨m,a⟩ ∈ r
  shows Minimum(r,A) = m
proof -
  from A2 A3 have I: HasAminimum(r,A) using HasAminimum_def
    by auto
  with A1 have ∃!m. m∈A ∧ (∀x∈A. ⟨m,x⟩ ∈ r)
    using Order_ZF_4_L2 by simp
  moreover from A2 A3 have m∈A ∧ (∀x∈A. ⟨m,x⟩ ∈ r) by simp
  moreover from A1 I have
    Minimum(r,A) ∈ A ∧ (∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r)
    using Order_ZF_4_L4 by simp
  ultimately show Minimum(r,A) = m by auto
qed
```

If a set does not have a maximum, then for any its element we can find one that is (strictly) greater.

```
lemma Order_ZF_4_L16:
  assumes A1: antisym(r) and A2: r {is total on} X and
  A3: A⊆X and
  A4: ¬HasAmaximum(r,A) and
  A5: x∈A
  shows ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x
proof -
  { assume A6: ∀y∈A. ⟨x,y⟩ ∉ r ∨ y=x
    have ∀y∈A. ⟨y,x⟩ ∈ r
    proof
      fix y assume A7: y∈A
```

```
      with A6 have ⟨x,y⟩ ∉ r ∨ y=x by simp
      with A2 A3 A5 A7 show ⟨y,x⟩ ∈ r
 using IsTotal_def Order_ZF_1_L1 by auto
    qed
    with A5 have ∃x∈A.∀y∈A. ⟨y,x⟩ ∈ r
      by auto
    with A4 have False using HasAmaximum_def by simp
  } then show ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x by auto
qed
```

7.2 Supremum and Infimum

In this section we consider the notions of supremum and infimum a set.

Elements of the set of upper bounds are indeed upper bounds. Isabelle also thinks it is obvious.

```
lemma Order_ZF_5_L1: assumes u ∈ (⋂a∈A. r{a}) and a∈A
  shows ⟨a,u⟩ ∈ r
  using assms by auto
```

Elements of the set of lower bounds are indeed lower bounds. Isabelle also thinks it is obvious.

```
lemma Order_ZF_5_L2: assumes l ∈ (⋂a∈A. r-{a}) and a∈A
  shows ⟨l,a⟩ ∈ r
  using assms by auto
```

If the set of upper bounds has a minimum, then the supremum is less or equal than any upper bound. We can probably do away with the assumption that A is not empty, (ab)using the fact that intersection over an empty family is defined in Isabelle to be empty. This lemma is obsolete and will be removed in the future. Use `sup_leq_up_bnd` instead.

```
lemma Order_ZF_5_L3: assumes A1: antisym(r) and A2: A≠0 and
  A3: HasAminimum(r,⋂a∈A. r{a}) and
  A4: ∀a∈A. ⟨a,u⟩ ∈ r
  shows ⟨Supremum(r,A),u⟩ ∈ r
proof -
  let U = ⋂a∈A. r{a}
  from A4 have ∀a∈A. u ∈ r{a} using image_singleton_iff
    by simp
  with A2 have u∈U by auto
  with A1 A3 show ⟨Supremum(r,A),u⟩ ∈ r
    using Order_ZF_4_L4 Supremum_def by simp
qed
```

Supremum is less or equal than any upper bound.

```
lemma sup_leq_up_bnd: assumes antisym(r) HasAsupremum(r,A) ∀a∈A. ⟨a,u⟩
∈ r
```

```
  shows ⟨Supremum(r,A),u⟩ ∈ r
proof -
  let U = ⋂a∈A. r{a}
  from assms(3) have  ∀a∈A. u ∈ r{a} using image_singleton_iff by simp
  with assms(2) have u∈U using set_sup_not_empty by auto
  with assms(1,2) show ⟨Supremum(r,A),u⟩ ∈ r
    unfolding HasAsupremum_def Supremum_def using Order_ZF_4_L4 by simp
qed
```

Infimum is greater or equal than any lower bound. This lemma is obsolete and will be removed. Use `inf_geq_lo_bnd` instead.

```
lemma Order_ZF_5_L4: assumes A1: antisym(r) and A2: A≠0 and
  A3: HasAmaximum(r,⋂a∈A. r-{a}) and
  A4: ∀a∈A. ⟨l,a⟩ ∈ r
  shows ⟨l,Infimum(r,A)⟩ ∈ r
proof -
  let L = ⋂a∈A. r-{a}
  from A4 have ∀a∈A. l ∈ r-{a} using vimage_singleton_iff
    by simp
  with A2 have l∈L by auto
  with A1 A3 show ⟨l,Infimum(r,A)⟩ ∈ r
    using Order_ZF_4_L3 Infimum_def by simp
qed
```

Infimum is greater or equal than any upper bound.

```
lemma inf_geq_lo_bnd: assumes antisym(r) HasAnInfimum(r,A) ∀a∈A. ⟨u,a⟩
∈ r
  shows ⟨u,Infimum(r,A)⟩ ∈ r
proof -
  let U = ⋂a∈A. r-{a}
  from assms(3) have  ∀a∈A. u ∈ r-{a} using vimage_singleton_iff by
simp
  with assms(2) have u∈U using set_inf_not_empty by auto
  with assms(1,2) show  ⟨u,Infimum(r,A)⟩ ∈ r
    unfolding HasAnInfimum_def Infimum_def using Order_ZF_4_L3 by simp
qed
```

If z is an upper bound for A and is less or equal than any other upper bound, then z is the supremum of A.

```
lemma Order_ZF_5_L5: assumes A1: antisym(r) and A2: A≠0 and
  A3: ∀x∈A. ⟨x,z⟩ ∈ r and
  A4: ∀y. (∀x∈A. ⟨x,y⟩ ∈ r) ⟶ ⟨z,y⟩ ∈ r
  shows
  HasAminimum(r,⋂a∈A. r{a})
  z = Supremum(r,A)
proof -
  let B = ⋂a∈A. r{a}
  from A2 A3 A4 have I: z ∈ B   ∀y∈B. ⟨z,y⟩ ∈ r
```

```
    by auto
  then show HasAminimum(r,⋂a∈A. r{a})
    using HasAminimum_def by auto
  from A1 I show z = Supremum(r,A)
    using Order_ZF_4_L15 Supremum_def by simp
qed
```

The dual theorem to `Order_ZF_5_L5`: if z is an lower bound for A and is greater or equal than any other lower bound, then z is the infimum of A.

```
lemma inf_glb:
  assumes antisym(r) A≠0 ∀x∈A. ⟨z,x⟩ ∈ r ∀y. (∀x∈A. ⟨y,x⟩ ∈ r) ⟶ ⟨y,z⟩
∈ r
  shows
  HasAmaximum(r,⋂a∈A. r-{a})
  z = Infimum(r,A)
proof -
  let B = ⋂a∈A. r-{a}
  from assms(2,3,4) have I: z ∈ B   ∀y∈B. ⟨y,z⟩ ∈ r
    by auto
  then show HasAmaximum(r,⋂a∈A. r-{a})
    unfolding HasAmaximum_def by auto
  from assms(1) I show z = Infimum(r,A)
    using Order_ZF_4_L14 Infimum_def by simp
qed
```

Supremum and infimum of a singleton is the element.

```
lemma sup_inf_singl: assumes antisym(r) refl(X,r) z∈X
  shows
    HasAsupremum(r,{z}) Supremum(r,{z}) = z and
    HasAnInfimum(r,{z}) Infimum(r,{z}) = z
proof -
  from assms show Supremum(r,{z}) = z and Infimum(r,{z}) = z
    using inf_glb Order_ZF_5_L5 unfolding refl_def by auto
  from assms show  HasAsupremum(r,{z})
    using Order_ZF_5_L5 unfolding HasAsupremum_def refl_def by blast
  from assms show HasAnInfimum(r,{z})
    using inf_glb unfolding HasAnInfimum_def refl_def by blast
qed
```

If a set has a maximum, then the maximum is the supremum. This lemma is obsolete, use `max_is_sup` instead.

```
lemma Order_ZF_5_L6:
  assumes A1:  antisym(r) and A2: A≠0 and
  A3: HasAmaximum(r,A)
  shows
  HasAminimum(r,⋂a∈A. r{a})
  Maximum(r,A) = Supremum(r,A)
proof -
```

```
  let M = Maximum(r,A)
  from A1 A3 have I: M ∈ A and II: ∀x∈A. ⟨x,M⟩ ∈ r
    using Order_ZF_4_L3 by auto
  from I have III: ∀y. (∀x∈A. ⟨x,y⟩ ∈ r) ⟶ ⟨M,y⟩ ∈ r
    by simp
  with A1 A2 II show HasAminimum(r,⋂a∈A. r{a})
    by (rule Order_ZF_5_L5)
  from A1 A2 II III show M = Supremum(r,A)
    by (rule Order_ZF_5_L5)
qed
```

Another version of `Order_ZF_5_L6` that: if a sat has a maximum then it has a supremum and the maximum is the supremum.

```
lemma max_is_sup: assumes antisym(r) A≠0 HasAmaximum(r,A)
  shows HasAsupremum(r,A) and Maximum(r,A) = Supremum(r,A)
proof -
  let M = Maximum(r,A)
  from assms(1,3) have M ∈ A and I: ∀x∈A. ⟨x,M⟩ ∈ r using Order_ZF_4_L3

    by auto
  with assms(1,2) have HasAminimum(r,⋂a∈A. r{a}) using Order_ZF_5_L5(1)

    by blast
  then show HasAsupremum(r,A) unfolding HasAsupremum_def by simp
  from assms(1,2) ‹M ∈ A› I show M = Supremum(r,A) using Order_ZF_5_L5(2)

    by blast
qed
```

Minimum is the infimum if it exists.

```
lemma min_is_inf: assumes antisym(r) A≠0 HasAminimum(r,A)
  shows HasAnInfimum(r,A) and Minimum(r,A) = Infimum(r,A)
proof -
  let M = Minimum(r,A)
  from assms(1,3) have M∈A and I: ∀x∈A. ⟨M,x⟩ ∈ r using  Order_ZF_4_L4

    by auto
  with assms(1,2) have HasAmaximum(r,⋂a∈A. r-{a}) using inf_glb(1) by
blast
  then show HasAnInfimum(r,A) unfolding HasAnInfimum_def by simp
  from assms(1,2) ‹M ∈ A› I show M = Infimum(r,A) using inf_glb(2) by
blast
qed
```

For reflexive and total relations two-element set has a minimum and a maximum.

```
lemma min_max_two_el: assumes r {is total on} X x∈X y∈X
  shows HasAminimum(r,{x,y}) and HasAmaximum(r,{x,y})
```

```
  using assms unfolding IsTotal_def HasAminimum_def HasAmaximum_def by
auto
```

For antisymmetric, reflexive and total relations two-element set has a supremum and infimum.

```
lemma inf_sup_two_el:assumes antisym(r) r {is total on} X x∈X y∈X
  shows
    HasAnInfimum(r,{x,y})
    Minimum(r,{x,y}) = Infimum(r,{x,y})
    HasAsupremum(r,{x,y})
    Maximum(r,{x,y}) = Supremum(r,{x,y})
  using assms min_max_two_el max_is_sup min_is_inf by auto
```

A sufficient condition for the supremum to be in the space.

```
lemma sup_in_space:
  assumes r ⊆ X×X antisym(r) HasAminimum(r,⋂a∈A. r{a})
  shows Supremum(r,A) ∈ X and ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
proof -
  from assms(3) have A≠0 using set_sup_not_empty unfolding HasAsupremum_def
by simp
  then obtain a where a∈A by auto
  with assms(1,2,3) show Supremum(r,A) ∈ X unfolding Supremum_def
    using Order_ZF_4_L4 Order_ZF_5_L1 by blast
  from assms(2,3) show ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r unfolding Supremum_def
    using Order_ZF_4_L4 by blast
qed
```

A sufficient condition for the infimum to be in the space.

```
lemma inf_in_space:
  assumes r ⊆ X×X antisym(r) HasAmaximum(r,⋂a∈A. r-{a})
  shows Infimum(r,A) ∈ X and ∀x∈A. ⟨Infimum(r,A),x⟩ ∈ r
proof -
  from assms(3) have A≠0 using set_inf_not_empty unfolding HasAnInfimum_def
by simp
  then obtain a where a∈A by auto
  with assms(1,2,3) show Infimum(r,A) ∈ X unfolding Infimum_def
    using Order_ZF_4_L3 Order_ZF_5_L1 by blast
  from assms(2,3) show ∀x∈A. ⟨Infimum(r,A),x⟩ ∈ r unfolding Infimum_def
    using Order_ZF_4_L3 by blast
qed
```

Properties of supremum of a set for complete relations.

```
lemma Order_ZF_5_L7:
  assumes A1: r ⊆ X×X and A2: antisym(r) and
  A3: r {is complete} and
  A4: A≠0 and A5: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r
  shows Supremum(r,A) ∈ X and ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
proof -
```

```
  from A3 A4 A5 have HasAminimum(r,⋂a∈A. r{a})
    unfolding IsBoundedAbove_def IsComplete_def by blast
  with A1 A2 show Supremum(r,A) ∈ X and ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
    using sup_in_space by auto
qed
```

Infimum of the set of infima of a collection of sets is infimum of the union.

```
lemma inf_inf:
  assumes
    r ⊆ X×X antisym(r) trans(r)
    ∀T∈𝒯. HasAnInfimum(r,T)
    HasAnInfimum(r,{Infimum(r,T).T∈𝒯})
  shows
    HasAnInfimum(r,⋃𝒯) and Infimum(r,{Infimum(r,T).T∈𝒯}) = Infimum(r,⋃𝒯)
proof -
  let i = Infimum(r,{Infimum(r,T).T∈𝒯})
  note assms(2)
  moreover from assms(4,5) have ⋃𝒯 ≠ 0 using set_inf_not_empty by
blast
  moreover
  have ∀T∈𝒯.∀t∈T. ⟨i,t⟩ ∈ r
  proof -
    { fix T t assume T∈𝒯 t∈T
      with assms(1,2,4) have ⟨Infimum(r,T),t⟩ ∈ r
        unfolding HasAnInfimum_def using inf_in_space(2) by blast
      moreover from assms(1,2,5) ‹T∈𝒯› have ⟨i,Infimum(r,T)⟩ ∈ r
        unfolding HasAnInfimum_def using inf_in_space(2) by blast
      moreover note assms(3)
      ultimately have ⟨i,t⟩ ∈ r unfolding trans_def by blast
    } thus thesis by simp
  qed
  hence I: ∀t∈⋃𝒯. ⟨i,t⟩ ∈ r by auto
  moreover have J: ∀y. (∀x∈⋃𝒯. ⟨y,x⟩ ∈ r) ⟶ ⟨y,i⟩ ∈ r
  proof -
    { fix y x assume A: ∀x∈⋃𝒯. ⟨y,x⟩ ∈ r
      with assms(2,4) have ∀a∈{Infimum(r,T).T∈𝒯}. ⟨y,a⟩ ∈ r using inf_geq_lo_bnd
        by simp
      with assms(2,5) have ⟨y,i⟩ ∈ r by (rule inf_geq_lo_bnd)
    } thus thesis by simp
  qed
  ultimately have HasAmaximum(r,⋂a∈⋃𝒯. r-{a}) by (rule inf_glb)
  then show HasAnInfimum(r,⋃𝒯) unfolding HasAnInfimum_def by simp
  from assms(2) ‹⋃𝒯 ≠ 0› I J show i = Infimum(r,⋃𝒯) by (rule inf_glb)
qed
```

Supremum of the set of suprema of a collection of sets is supremum of the union.

```
lemma sup_sup:
  assumes
```

```
    r ⊆ X×X antisym(r) trans(r)
    ∀T∈𝒯. HasAsupremum(r,T)
    HasAsupremum(r,{Supremum(r,T).T∈𝒯})
  shows
    HasAsupremum(r,⋃𝒯) and Supremum(r,{Supremum(r,T).T∈𝒯}) = Supremum(r,⋃𝒯)
proof -
  let s = Supremum(r,{Supremum(r,T).T∈𝒯})
  note assms(2)
  moreover from assms(4,5) have ⋃𝒯 ≠ 0 using set_sup_not_empty by
blast
  moreover
  have ∀T∈𝒯.∀t∈T. ⟨t,s⟩ ∈ r
  proof -
    { fix T t assume T∈𝒯 t∈T
      with assms(1,2,4) have ⟨t,Supremum(r,T)⟩ ∈ r
        unfolding HasAsupremum_def using sup_in_space(2) by blast
      moreover from assms(1,2,5) ‹T∈𝒯› have ⟨Supremum(r,T),s⟩ ∈ r
        unfolding HasAsupremum_def using sup_in_space(2) by blast
      moreover note assms(3)
      ultimately have ⟨t,s⟩ ∈ r unfolding trans_def by blast
    } thus thesis by simp
  qed
  hence I: ∀t∈⋃𝒯. ⟨t,s⟩ ∈ r by auto
  moreover have J: ∀y. (∀x∈⋃𝒯. ⟨x,y⟩ ∈ r) ⟶ ⟨s,y⟩ ∈ r
  proof -
    { fix y x assume A: ∀x∈⋃𝒯. ⟨x,y⟩ ∈ r
      with assms(2,4) have ∀a∈{Supremum(r,T).T∈𝒯}. ⟨a,y⟩ ∈ r using sup_leq_up_bnd
        by simp
      with assms(2,5) have ⟨s,y⟩ ∈ r by (rule sup_leq_up_bnd)
    } thus thesis by simp
  qed
  ultimately have HasAminimum(r,⋂a∈⋃𝒯. r{a}) by (rule Order_ZF_5_L5)
  then show HasAsupremum(r,⋃𝒯) unfolding HasAsupremum_def by simp
  from assms(2) ‹⋃𝒯 ≠ 0› I J show s = Supremum(r,⋃𝒯) by (rule Order_ZF_5_L5)
qed
```

If the relation is a linear order then for any element y smaller than the supremum of a set we can find one element of the set that is greater than y.

```
lemma Order_ZF_5_L8:
  assumes A1: r ⊆ X×X  and A2: IsLinOrder(X,r) and
  A3: r {is complete} and
  A4: A⊆X  A≠0 and A5: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r and
  A6: ⟨y,Supremum(r,A)⟩ ∈ r    y ≠ Supremum(r,A)
  shows ∃z∈A. ⟨y,z⟩ ∈ r ∧ y ≠ z
proof -
  from A2 have
    I: antisym(r) and
    II: trans(r) and
    III: r {is total on} X
```

```
    using IsLinOrder_def by auto
  from A1 A6 have T1: y∈X by auto
  { assume A7: ∀z ∈ A. ⟨y,z⟩ ∉ r ∨ y=z
    from A4 I have antisym(r) and A≠0 by auto
    moreover have ∀x∈A. ⟨x,y⟩ ∈ r
    proof
      fix x assume A8: x∈A
      with A4 have T2: x∈X by auto
      from A7 A8 have ⟨y,x⟩ ∉ r ∨ y=x by simp
      with III T1 T2 show ⟨x,y⟩ ∈ r
 using IsTotal_def total_is_refl refl_def by auto
    qed
    moreover have ∀u. (∀x∈A. ⟨x,u⟩ ∈ r) ⟶ ⟨y,u⟩ ∈ r
    proof-
      { fix u assume A9: ∀x∈A. ⟨x,u⟩ ∈ r
 from A4 A5 have IsBoundedAbove(A,r) and A≠0
   using IsBoundedAbove_def by auto
 with  A3 A4 A6 I A9  have
   ⟨y,Supremum(r,A)⟩ ∈ r ∧ ⟨Supremum(r,A),u⟩ ∈ r
   using IsComplete_def Order_ZF_5_L3 by simp
 with II have ⟨y,u⟩ ∈ r by (rule Fol1_L3)
      } then show ∀u. (∀x∈A. ⟨x,u⟩ ∈ r) ⟶ ⟨y,u⟩ ∈ r
 by simp
    qed
    ultimately have y = Supremum(r,A)
      by (rule Order_ZF_5_L5)
    with A6 have False by simp
  } then show ∃z∈A. ⟨y,z⟩ ∈ r ∧ y ≠ z by auto
qed
```

7.3 Strict versions of order relations

One of the problems with translating formalized mathematics from Metamath to IsarMathLib is that Metamath uses strict orders (of the $<$ type) while in IsarMathLib we mostly use nonstrict orders (of the $\leq$ type). This doesn't really make any difference, but is annoying as we have to prove many theorems twice. In this section we prove some theorems to make it easier to translate the statements about strict orders to statements about the corresponding non-strict order and vice versa.

We define a strict version of a relation by removing the $y = x$ line from the relation.

```
definition
  StrictVersion(r) ≡ r - {⟨x,x⟩. x ∈ domain(r)}
```

A reformulation of the definition of a strict version of an order.

```
lemma def_of_strict_ver: shows
  ⟨x,y⟩ ∈ StrictVersion(r) ⟷ ⟨x,y⟩ ∈ r ∧ x≠y
```

```
  using StrictVersion_def domain_def by auto
```

The next lemma is about the strict version of an antisymmetric relation.

```
lemma strict_of_antisym:
  assumes A1: antisym(r) and A2: ⟨a,b⟩ ∈ StrictVersion(r)
  shows ⟨b,a⟩ ∉ StrictVersion(r)
proof -
  { assume A3: ⟨b,a⟩ ∈ StrictVersion(r)
    with A2 have ⟨a,b⟩ ∈ r  and ⟨b,a⟩ ∈ r
      using def_of_strict_ver by auto
    with A1 have a=b by (rule Fol1_L4)
    with A2 have False using def_of_strict_ver
      by simp
  } then show ⟨b,a⟩ ∉ StrictVersion(r) by auto
qed
```

The strict version of totality.

```
lemma strict_of_tot:
  assumes r {is total on} X and a∈X  b∈X  a≠b
  shows ⟨a,b⟩ ∈ StrictVersion(r) ∨ ⟨b,a⟩ ∈ StrictVersion(r)
  using assms IsTotal_def def_of_strict_ver by auto
```

A trichotomy law for the strict version of a total and antisymmetric relation. It is kind of interesting that one does not need the full linear order for this.

```
lemma strict_ans_tot_trich:
  assumes A1: antisym(r) and A2: r {is total on} X
  and A3: a∈X  b∈X
  and A4: s = StrictVersion(r)
  shows Exactly_1_of_3_holds(⟨a,b⟩ ∈ s, a=b,⟨b,a⟩ ∈ s)
proof -
  let p = ⟨a,b⟩ ∈ s
  let q = a=b
  let r = ⟨b,a⟩ ∈ s
  from A2 A3 A4 have p ∨ q ∨ r
    using strict_of_tot by auto
  moreover from A1 A4 have p ⟶ ¬q ∧ ¬r
    using def_of_strict_ver strict_of_antisym by simp
  moreover from A4 have q ⟶ ¬p ∧ ¬r
    using def_of_strict_ver by simp
  moreover from A1 A4 have r ⟶ ¬p ∧ ¬q
    using def_of_strict_ver strict_of_antisym by auto
  ultimately show Exactly_1_of_3_holds(p, q, r)
    by (rule Fol1_L5)
qed
```

A trichotomy law for linear order. This is a special case of `strict_ans_tot_trich`.

```
corollary strict_lin_trich: assumes A1: IsLinOrder(X,r) and
  A2: a∈X  b∈X and
```

```
  A3: s = StrictVersion(r)
  shows Exactly_1_of_3_holds(⟨a,b⟩ ∈ s, a=b,⟨b,a⟩ ∈ s)
  using assms IsLinOrder_def strict_ans_tot_trich by auto
```

For an antisymmetric relation if a pair is in relation then the reversed pair is not in the strict version of the relation.

```
lemma geq_impl_not_less:
  assumes A1: antisym(r) and A2: ⟨a,b⟩ ∈ r
  shows ⟨b,a⟩ ∉ StrictVersion(r)
proof -
  { assume A3: ⟨b,a⟩ ∈  StrictVersion(r)
    with A2 have ⟨a,b⟩ ∈ StrictVersion(r)
      using def_of_strict_ver by auto
    with A1 A3 have False using strict_of_antisym
      by blast
  } then show ⟨b,a⟩ ∉ StrictVersion(r) by auto
qed
```

If an antisymmetric relation is transitive, then the strict version is also transitive, an explicit version `strict_of_transB` below.

```
lemma strict_of_transA:
  assumes A1: trans(r) and A2: antisym(r) and
  A3: s= StrictVersion(r) and  A4: ⟨a,b⟩ ∈ s  ⟨b,c⟩ ∈ s
  shows ⟨a,c⟩ ∈ s
proof -
  from A3 A4 have I: ⟨a,b⟩ ∈ r ∧ ⟨b,c⟩ ∈ r
    using def_of_strict_ver by simp
  with A1 have ⟨a,c⟩ ∈ r by (rule Fol1_L3)
  moreover
  { assume a=c
    with I have ⟨a,b⟩ ∈ r and ⟨b,a⟩ ∈ r by auto
    with A2 have a=b by (rule Fol1_L4)
    with A3 A4 have False using def_of_strict_ver by simp
  } then have a≠c by auto
  ultimately have  ⟨a,c⟩ ∈ StrictVersion(r)
    using def_of_strict_ver by simp
  with A3 show thesis by simp
qed
```

If an antisymmetric relation is transitive, then the strict version is also transitive.

```
lemma strict_of_transB:
  assumes A1: trans(r) and A2: antisym(r)
  shows trans(StrictVersion(r))
proof -
  let s = StrictVersion(r)
  from A1 A2 have
    ∀ x y z. ⟨x, y⟩ ∈ s ∧ ⟨y, z⟩ ∈ s ⟶ ⟨x, z⟩ ∈ s
```

```
    using strict_of_transA by blast
  then show trans(StrictVersion(r)) by (rule Fol1_L2)
qed
```

The next lemma provides a condition that is satisfied by the strict version of a relation if the original relation is a complete linear order.

```
lemma strict_of_compl:
  assumes A1: r ⊆ X×X and A2: IsLinOrder(X,r) and
  A3: r {is complete} and
  A4: A⊆X  A≠0 and A5: s = StrictVersion(r) and
  A6: ∃u∈X. ∀y∈A. ⟨y,u⟩ ∈ s
  shows
  ∃x∈X. ( ∀y∈A. ⟨x,y⟩ ∉ s ) ∧ (∀y∈X. ⟨y,x⟩ ∈ s ⟶ (∃z∈A. ⟨y,z⟩ ∈ s))
proof -
  let x = Supremum(r,A)
  from A2 have I: antisym(r) using IsLinOrder_def
    by simp
  moreover from A5 A6 have ∃u∈X. ∀y∈A. ⟨y,u⟩ ∈ r
    using def_of_strict_ver by auto
  moreover note A1 A3 A4
  ultimately have II: x ∈ X   ∀y∈A. ⟨y,x⟩ ∈ r
    using Order_ZF_5_L7 by auto
  then have III: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r by auto
  from A5 I II have x ∈ X   ∀y∈A. ⟨x,y⟩ ∉ s
    using geq_impl_not_less by auto
  moreover from A1 A2 A3 A4 A5 III have
    ∀y∈X. ⟨y,x⟩ ∈ s ⟶ (∃z∈A. ⟨y,z⟩ ∈ s)
    using def_of_strict_ver Order_ZF_5_L8 by simp
  ultimately show
    ∃x∈X. ( ∀y∈A. ⟨x,y⟩ ∉ s ) ∧ (∀y∈X. ⟨y,x⟩ ∈ s ⟶ (∃z∈A. ⟨y,z⟩ ∈
s))
    by auto
qed
```

Strict version of a relation on a set is a relation on that set.

```
lemma strict_ver_rel: assumes A1: r ⊆ A×A
  shows StrictVersion(r) ⊆ A×A
  using assms StrictVersion_def by auto

end
```

8 Order on natural numbers

```
theory NatOrder_ZF imports Nat_ZF_IML Order_ZF

begin
```

This theory proves that $\leq$ is a linear order on $\mathbb{N}$. $\leq$ is defined in Isabelle's

`Nat` theory, and linear order is defined in `Order_ZF` theory. Contributed by Seo Sanghyeon.

8.1 Order on natural numbers

This is the only section in this theory.

If a, b are natural numbers then a is less or equal b or b is (strictly) less than a. We use a result on ordinals in the proof.

```
lemma nat_order_2cases:  assumes a∈nat and b∈nat
  shows a ≤ b ∨ b < a
proof -
   from assms have I: Ord(a) ∧ Ord(b)
    using nat_into_Ord by auto
  then have a ∈ b ∨ a = b ∨ b ∈ a
    using Ord_linear by simp
  with I have a < b ∨ a = b ∨ b < a
    using ltI by auto
  with I show a ≤ b ∨ b < a
    using le_iff by auto
qed
```

A special case of `nat_order_2cases`: If a, b are natural numbers then a is less or equal b or b is less or equal than a.

```
lemma NatOrder_ZF_1_L1:
  assumes a∈nat and b∈nat
  shows a ≤ b ∨ b ≤ a
  using assms nat_order_2cases leI by auto
```

$\leq$ is antisymmetric, transitive, total, and linear. Proofs by rewrite using definitions.

```
lemma NatOrder_ZF_1_L2:
  shows
  antisym(Le)
  trans(Le)
  Le {is total on} nat
  IsLinOrder(nat,Le)
proof -
  show antisym(Le)
    using antisym_def Le_def le_anti_sym by auto
  moreover show trans(Le)
    using trans_def Le_def le_trans by blast
  moreover show Le {is total on} nat
    using IsTotal_def Le_def NatOrder_ZF_1_L1 by simp
  ultimately show IsLinOrder(nat,Le)
    using IsLinOrder_def by simp
qed
```

The order on natural numbers is linear on every natural number. Recall that each natural number is a subset of the set of all natural numbers (as well as a member).

```
lemma natord_lin_on_each_nat:
  assumes A1: n ∈ nat shows IsLinOrder(n,Le)
proof -
  from A1 have n ⊆ nat using nat_subset_nat
    by simp
  then show thesis using NatOrder_ZF_1_L2 ord_linear_subset
    by blast
qed
```

end

9 Functions - introduction

```
theory func1 imports ZF.func Fol1 ZF1
```

begin

This theory covers basic properties of function spaces. A set of functions with domain X and values in the set Y is denoted in Isabelle as $X \to Y$. It just happens that the colon ":" is a synonym of the set membership symbol $\in$ in Isabelle/ZF so we can write $f : X \to Y$ instead of $f \in X \to Y$. This is the only case that we use the colon instead of the regular set membership symbol.

9.1 Properties of functions, function spaces and (inverse) images.

Functions in ZF are sets of pairs. This means that if $f : X \to Y$ then $f \subseteq X \times Y$. This section is mostly about consequences of this understanding of the notion of function.

We define the notion of function that preserves a collection here. Given two collection of sets a function preserves the collections if the inverse image of sets in one collection belongs to the second one. This notion does not have a name in romantic math. It is used to define continuous functions in `Topology_ZF_2` theory. We define it here so that we can use it for other purposes, like defining measurable functions. Recall that `f-(A)` means the inverse image of the set A.

```
definition
  PresColl(f,S,T) ≡ ∀ A∈T. f-(A)∈S
```

A definition that allows to get the first factor of the domain of a binary function $f : X \times Y \to Z$.

```
definition
  fstdom(f) ≡ domain(domain(f))
```

If a function maps A into another set, then A is the domain of the function.

```
lemma func1_1_L1: assumes f:A→C shows domain(f) = A
  using assms domain_of_fun by simp
```

Standard Isabelle defines a `function(f)` predicate. The next lemma shows that our functions satisfy that predicate. It is a special version of Isabelle's `fun_is_function`.

```
lemma fun_is_fun: assumes f:X→Y shows function(f)
  using assms fun_is_function by simp
```

A lemma explains what `fstdom` is for.

```
lemma fstdomdef: assumes A1: f: X×Y → Z and A2: Y≠∅
  shows fstdom(f) = X
proof -
  from A1 have domain(f) = X×Y using func1_1_L1
    by simp
  with A2 show fstdom(f) = X unfolding fstdom_def by auto
qed
```

A version of the `Pi_type` lemma from the standard Isabelle/ZF library.

```
lemma func1_1_L1A: assumes A1: f:X→Y and A2: ∀x∈X. f(x) ∈ Z
  shows f:X→Z
proof -
  { fix x assume x∈X
    with A2 have f(x) ∈ Z by simp }
  with A1 show f:X→Z by (rule Pi_type)
qed
```

A variant of `func1_1_L1A`.

```
lemma func1_1_L1B: assumes A1: f:X→Y and A2: Y⊆Z
  shows f:X→Z
proof -
  from A1 A2 have ∀x∈X. f(x) ∈ Z
    using apply_funtype by auto
  with A1 show  f:X→Z using func1_1_L1A by blast
qed
```

There is a value for each argument.

```
lemma func1_1_L2: assumes A1: f:X→Y  x∈X
  shows ∃y∈Y. ⟨x,y⟩ ∈ f
proof -
  from A1 have f(x) ∈ Y using apply_type by simp
  moreover from A1 have ⟨ x,f(x)⟩∈ f using apply_Pair by simp
  ultimately show thesis by auto
```

```
qed
```

The inverse image is the image of converse. True for relations as well.

```
lemma vimage_converse: shows r-(A) = converse(r)(A)
  using vimage_iff image_iff converse_iff by auto
```

The image is the inverse image of converse.

```
lemma image_converse: shows converse(r)-(A) = r(A)
  using vimage_iff image_iff converse_iff by auto
```

The inverse image by a composition is the composition of inverse images.

```
lemma vimage_comp: shows (r O s)-(A) = s-(r-(A))
  using vimage_converse converse_comp image_comp image_converse by simp
```

A version of `vimage_comp` for three functions.

```
lemma vimage_comp3: shows (r O s O t)-(A) = t-(s-(r-(A)))
  using vimage_comp by simp
```

Inverse image of any set is contained in the domain.

```
lemma func1_1_L3: assumes A1: f:X→Y shows f-(D) ⊆ X
proof -
   have ∀x. x∈f-(D) ⟶ x ∈ domain(f)
      using  vimage_iff domain_iff by auto
    with A1 have ∀x. (x ∈ f-(D)) ⟶ (x∈X) using func1_1_L1 by simp
    then show thesis by auto
qed
```

The inverse image of the range is the domain.

```
lemma func1_1_L4: assumes f:X→Y shows f-(Y) = X
  using assms func1_1_L3 func1_1_L2 vimage_iff by blast
```

The arguments belongs to the domain and values to the range.

```
lemma func1_1_L5:
  assumes A1: ⟨ x,y⟩ ∈ f and A2: f:X→Y
  shows x∈X ∧ y∈Y
proof
  from A1 A2 show x∈X using apply_iff by simp
  with A2 have f(x)∈ Y using apply_type by simp
  with A1 A2 show y∈Y using apply_iff by simp
qed
```

Function is a subset of cartesian product.

```
lemma fun_subset_prod: assumes A1: f:X→Y shows f ⊆ X×Y
proof
  fix p assume p ∈ f
  with A1 have ∃x∈X. p = ⟨x, f(x)⟩
    using Pi_memberD by simp
```

```
  then obtain x where I: p = ⟨x, f(x)⟩
    by auto
  with A1 ‹p ∈ f› have x∈X ∧ f(x) ∈ Y
    using func1_1_L5 by blast
  with I show p ∈ X×Y by auto
qed
```

The (argument, value) pair belongs to the graph of the function.

```
lemma func1_1_L5A:
  assumes A1: f:X→Y  x∈X  y = f(x)
  shows ⟨x,y⟩ ∈ f  y ∈ range(f)
proof -
  from A1 show ⟨x,y⟩ ∈ f using apply_Pair by simp
  then show y ∈ range(f) using rangeI by simp
qed
```

The next theorem illustrates the meaning of the concept of function in ZF.

```
theorem fun_is_set_of_pairs: assumes A1: f:X→Y
  shows f = {⟨x,f(x)⟩. x ∈ X}
proof
  from A1 show {⟨x, f(x)⟩. x ∈ X} ⊆ f using func1_1_L5A
    by auto
next
  { fix p assume p ∈ f
    with A1 have p ∈ X×Y using fun_subset_prod
      by auto
    with A1 ‹p ∈ f› have p ∈ {⟨x, f(x)⟩. x ∈ X}
      using apply_equality by auto
  } thus f ⊆ {⟨x, f(x)⟩. x ∈ X} by auto
qed
```

If a pair $\langle x, y\rangle$ is a member of a function f, then x is in the domain and $y = f(x)$.

```
lemma pair_fun_member: assumes f:X→Y and ⟨x,y⟩ ∈ f
  shows x∈X and y=f(x)
proof -
  from assms have ⟨x,y⟩ ∈ {⟨x,f(x)⟩. x ∈ X} using fun_is_set_of_pairs
    by simp
  then show x∈X and y=f(x) by auto
qed
```

The range of function that maps X into Y is contained in Y.

```
lemma func1_1_L5B:
  assumes  A1: f:X→Y shows range(f) ⊆ Y
proof
  fix y assume y ∈ range(f)
  then obtain x where ⟨ x,y⟩ ∈ f
    using range_def converse_def domain_def by auto
```

```
  with A1 show y∈Y using func1_1_L5 by blast
qed
```

The image of any set is contained in the range.

```
lemma func1_1_L6: assumes A1: f:X→Y
  shows f(B) ⊆ range(f) and f(B) ⊆ Y
proof -
  show f(B) ⊆ range(f) using image_iff rangeI by auto
  with A1 show f(B) ⊆ Y using func1_1_L5B by blast
qed
```

The inverse image of any set is contained in the domain.

```
lemma func1_1_L6A: assumes A1: f:X→Y shows f-(A)⊆X
proof
  fix x
  assume A2: x∈f-(A) then obtain y where ⟨x,y⟩ ∈ f
    using vimage_iff by auto
  with A1 show  x∈X using func1_1_L5 by fast
qed
```

Image of a greater set is greater.

```
lemma func1_1_L8: assumes A1: A⊆B  shows f(A)⊆ f(B)
  using assms image_Un by auto
```

An immediate corollary of `vimage_mono` from the Isabelle/ZF distribution - the inverse image of a greater set is greater. Note we do not require that f is a function, so this is true for relations as well.

```
lemma vimage_mono1: assumes A⊆B shows f-(A) ⊆ f-(B)
  using assms vimage_mono by simp
```

A set is contained in the the inverse image of its image. There is similar theorem in `equalities.thy` (`function_image_vimage`) which shows that the image of inverse image of a set is contained in the set.

```
lemma func1_1_L9: assumes A1: f:X→Y and A2: A⊆X
  shows A ⊆ f-(f(A))
proof -
  from A1 A2 have ∀x∈A. ⟨ x,f(x)⟩ ∈ f  using apply_Pair by auto
  then show thesis using image_iff by auto
qed
```

The inverse image of the image of the domain is the domain.

```
lemma inv_im_dom: assumes A1: f:X→Y shows f-(f(X)) = X
proof
  from A1 show f-(f(X)) ⊆ X using func1_1_L3 by simp
  from A1 show X ⊆ f-(f(X)) using func1_1_L9 by simp
qed
```

A technical lemma needed to make the `func1_1_L11` proof more clear.

```
lemma func1_1_L10:
  assumes A1: f ⊆ X×Y and A2: ∃!y. (y∈Y ∧ ⟨x,y⟩ ∈ f)
  shows ∃!y. ⟨x,y⟩ ∈ f
proof
  from A2 show ∃y. ⟨x, y⟩ ∈ f by auto
  fix y n assume ⟨x,y⟩ ∈ f and ⟨x,n⟩ ∈ f
  with A1 A2 show y=n by auto
qed
```

If $f \subseteq X \times Y$ and for every $x \in X$ there is exactly one $y \in Y$ such that $(x, y) \in f$ then f maps X to Y.

```
lemma func1_1_L11:
  assumes f ⊆ X×Y and ∀x∈X. ∃!y. y∈Y ∧ ⟨x,y⟩ ∈ f
  shows f: X→Y using assms func1_1_L10 Pi_iff_old by simp
```

A set defined by a lambda-type expression is a function. There is a similar lemma in func.thy, but I had problems with lambda expressions syntax so I could not apply it. This lemma is a workaround for this. Besides, lambda expressions are not readable.

```
lemma func1_1_L11A: assumes A1: ∀x∈X. b(x) ∈ Y
  shows {⟨x,y⟩ ∈ X×Y. b(x) = y} : X→Y
proof -
  let f = {⟨ x,y⟩ ∈ X×Y. b(x) = y}
  have f ⊆ X×Y by auto
  moreover have ∀x∈X. ∃!y. y∈Y ∧ ⟨ x,y⟩ ∈ f
  proof
    fix x assume A2: x∈X
    show ∃!y. y∈Y ∧ ⟨x, y⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
    proof
      from A2 A1 show
        ∃y. y∈Y ∧ ⟨x, y⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
 by simp
    next
      fix y y1
      assume y∈Y ∧ ⟨x, y⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
 and y1∈Y ∧ ⟨x, y1⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
      then show y = y1 by simp
    qed
  qed
  ultimately show {⟨ x,y⟩ ∈ X×Y. b(x) = y} : X→Y
    using func1_1_L11 by simp
qed
```

The next lemma will replace `func1_1_L11A` one day.

```
lemma ZF_fun_from_total: assumes A1: ∀x∈X. b(x) ∈ Y
  shows {⟨x,b(x)⟩. x∈X} : X→Y
proof -
  let f = {⟨x,b(x)⟩. x∈X}
```

```
  { fix x assume A2: x∈X
    have ∃!y. y∈Y ∧ ⟨x, y⟩ ∈ f
    proof
 from A1 A2 show ∃y. y∈Y ∧ ⟨x, y⟩ ∈ f
 by simp
    next fix y y1 assume y∈Y ∧ ⟨x, y⟩ ∈ f
 and y1∈Y ∧ ⟨x, y1⟩ ∈ f
      then show y = y1 by simp
    qed
  } then have ∀x∈X. ∃!y. y∈Y ∧ ⟨ x,y⟩ ∈ f
    by simp
  moreover from A1 have f ⊆ X×Y by auto
  ultimately show thesis using func1_1_L11
    by simp
qed
```

The value of a function defined by a meta-function is this meta-function (deprecated, use `ZF_fun_from_tot_val(1)` instead).

```
lemma func1_1_L11B:
  assumes A1: f:X→Y   x∈X
  and A2: f = {⟨x,y⟩ ∈ X×Y. b(x) = y}
  shows f(x) = b(x)
proof -
  from A1 have ⟨ x,f(x)⟩ ∈ f using apply_iff by simp
  with A2 show thesis by simp
qed
```

The next lemma will replace `func1_1_L11B` one day.

```
lemma ZF_fun_from_tot_val:
  assumes f:X→Y   x∈X
  and f = {⟨x,b(x)⟩. x∈X}
  shows f(x) = b(x) and b(x)∈Y
proof -
  from assms(1,2) have ⟨x,f(x)⟩ ∈ f using apply_iff by simp
  with assms(3) show f(x) = b(x) by simp
  from assms(1,2) have f(x)∈Y by (rule apply_funtype)
  with ‹f(x) = b(x)› show b(x)∈Y by simp
qed
```

Identical meaning as `ZF_fun_from_tot_val`, but phrased a bit differently.

```
lemma ZF_fun_from_tot_val0:
  assumes f:X→Y and f = {⟨x,b(x)⟩. x∈X}
  shows ∀x∈X. f(x) = b(x)
  using assms ZF_fun_from_tot_val by simp
```

Another way of expressing that lambda expression is a function.

```
lemma lam_is_fun_range: assumes f={⟨x,g(x)⟩. x∈X}
  shows f:X→range(f)
```

```
proof -
  have ∀x∈X. g(x) ∈ range({⟨x,g(x)⟩. x∈X}) unfolding range_def
    by auto
  then have {⟨x,g(x)⟩. x∈X} : X→range({⟨x,g(x)⟩. x∈X}) by (rule ZF_fun_from_total)
  with assms show thesis by auto
qed
```

Yet another way of expressing value of a function.

```
lemma ZF_fun_from_tot_val1:
  assumes x∈X shows {⟨x,b(x)⟩. x∈X}(x)=b(x)
proof -
  let f = {⟨x,b(x)⟩. x∈X}
  have f:X→range(f) using lam_is_fun_range by simp
  with assms show thesis using ZF_fun_from_tot_val0 by simp
qed
```

An hypotheses-free form of ZF_fun_from_tot_val1: the value of a function $X \ni x \mapsto p(x)$ is $p(x)$ for all $x \in X$.

```
lemma ZF_fun_from_tot_val2: shows ∀x∈X. {⟨x,b(x)⟩. x∈X}(x) = b(x)
  using ZF_fun_from_tot_val1 by simp
```

The range of a function defined by set comprehension is the set of its values.

```
lemma range_fun: shows range({⟨x,b(x)⟩. x∈X}) = {b(x). x∈X}
  by blast
```

In Isabelle/ZF and Metamath if x is not in the domain of a function f then $f(x)$ is the empty set. This allows us to conclude that if $y \in f(x)$, then x must be en element of the domain of f.

```
lemma arg_in_domain: assumes f:X→Y y∈f(x) shows x∈X
proof -
  { assume x∉X
    with assms have False using func1_1_L1 apply_0 by simp
  } thus thesis by auto
qed
```

If x is not in the domain of the function then both the image of the singleton $\{x\}$ and the value of the function are empty. The second of the assertions is also proven by standard Isabelle/ZF apply_0 lemma in the func theory.

```
lemma arg_not_in_domain: assumes f:X→Y and x∉X
  shows f{x} = ∅ and f(x) = ∅
proof -
  from assms show f{x} = ∅ using func1_1_L1 by blast
  then show f(x) = ∅ unfolding apply_def by simp
qed
```

We can extend a function by specifying its values on a set disjoint with the domain.

```
lemma func1_1_L11C: assumes A1: f:X→Y and A2: ∀x∈A. b(x)∈B
  and A3: X∩A = ∅ and Dg: g = f ∪ {⟨x,b(x)⟩. x∈A}
  shows
  g : X∪A → Y∪B
  ∀x∈X. g(x) = f(x)
  ∀x∈A. g(x) = b(x)
proof -
  let h = {⟨x,b(x)⟩. x∈A}
  from A1 A2 A3 have
    I: f:X→Y  h : A→B  X∩A = ∅
    using ZF_fun_from_total by auto
  then have f∪h : X∪A → Y∪B
    by (rule fun_disjoint_Un)
  with Dg show g : X∪A → Y∪B by simp
  { fix x assume A4: x∈A
    with A1 A3 have (f∪h)(x) = h(x)
      using func1_1_L1 fun_disjoint_apply2
      by blast
    moreover from I A4 have h(x) = b(x)
      using ZF_fun_from_tot_val by simp
    ultimately have (f∪h)(x) = b(x)
      by simp
  } with Dg show ∀x∈A. g(x) = b(x) by simp
  { fix x assume A5: x∈X
    with A3 I have x ∉ domain(h)
      using func1_1_L1 by auto
    then have (f∪h)(x) = f(x)
      using fun_disjoint_apply1 by simp
  } with Dg show ∀x∈X. g(x) = f(x) by simp
qed
```

We can extend a function by specifying its value at a point that does not belong to the domain.

```
lemma func1_1_L11D: assumes A1: f:X→Y and A2: a∉X
  and Dg: g = f ∪ {⟨a,b⟩}
  shows
  g : X∪{a} → Y∪{b}
  ∀x∈X. g(x) = f(x)
  g(a) = b
proof -
  let h = {⟨a,b⟩}
  from A1 A2 Dg have I:
    f:X→Y  ∀x∈{a}. b∈{b}  X∩{a} = ∅  g = f ∪ {⟨x,b⟩. x∈{a}}
    by auto
  then show g : X∪{a} → Y∪{b}
    by (rule func1_1_L11C)
  from I show ∀x∈X. g(x) = f(x)
    by (rule func1_1_L11C)
  from I have ∀x∈{a}. g(x) = b
```

```
    by (rule func1_1_L11C)
  then show g(a) = b by auto
qed
```

A technical lemma about extending a function both by defining on a set disjoint with the domain and on a point that does not belong to any of those sets.

```
lemma func1_1_L11E:
  assumes A1: f:X→Y and
  A2: ∀x∈A. b(x)∈B and
  A3: X∩A = ∅ and A4: a ∉ X∪A
  and Dg: g = f ∪ {⟨x,b(x)⟩. x∈A} ∪ {⟨a,c⟩}
  shows
  g : X∪A∪{a} → Y∪B∪{c}
  ∀x∈X. g(x) = f(x)
  ∀x∈A. g(x) = b(x)
  g(a) = c
proof -
  let h = f ∪ {⟨x,b(x)⟩. x∈A}
  from assms show g : X∪A∪{a} → Y∪B∪{c}
    using func1_1_L11C func1_1_L11D by simp
  from A1 A2 A3 have I:
    f:X→Y  ∀x∈A. b(x)∈B  X∩A = ∅  h = f ∪ {⟨x,b(x)⟩. x∈A}
    by auto
  from assms have
    II: h : X∪A → Y∪B  a∉ X∪A  g = h ∪ {⟨a,c⟩}
    using func1_1_L11C by auto
  then have III: ∀x∈X∪A. g(x) = h(x) by (rule func1_1_L11D)
  moreover from I have  ∀x∈X. h(x) = f(x)
    by (rule func1_1_L11C)
  ultimately show ∀x∈X. g(x) = f(x) by simp
  from I have ∀x∈A. h(x) = b(x) by (rule func1_1_L11C)
  with III show ∀x∈A. g(x) = b(x) by simp
  from II show g(a) = c by (rule func1_1_L11D)
qed
```

A way of defining a function on a union of two possibly overlapping sets. We decompose the union into two differences and the intersection and define a function separately on each part.

```
lemma fun_union_overlap: assumes ∀x∈A∩B. h(x) ∈ Y  ∀x∈A\B. f(x) ∈
Y  ∀x∈B\A. g(x) ∈ Y
  shows {⟨x,if x∈A\B then f(x) else if x∈B\A then g(x) else h(x)⟩. x ∈
A∪B}: A∪B → Y
proof -
  let F = {⟨x,if x∈A\B then f(x) else if x∈B\A then g(x) else h(x)⟩. x
∈ A∩B}
  from assms have ∀x∈A∪B. (if x∈A\B then f(x) else if x∈B\A then g(x)
else h(x)) ∈ Y
    by auto
```

```
  then show thesis by (rule ZF_fun_from_total)
qed
```

Inverse image of intersection is the intersection of inverse images.

```
lemma invim_inter_inter_invim: assumes f:X→Y
  shows f-(A∩B) = f-(A) ∩ f-(B)
  using assms fun_is_fun function_vimage_Int by simp
```

The inverse image of an intersection of a nonempty collection of sets is the intersection of the inverse images. This generalizes `invim_inter_inter_invim` which is proven for the case of two sets.

```
lemma func1_1_L12:
  assumes A1: B ⊆ Pow(Y) and A2: B≠∅ and A3: f:X→Y
  shows f-(⋂B) = (⋂U∈B. f-(U))
proof
  from A2 show  f-(⋂B) ⊆ (⋂U∈B. f-(U)) by blast
  show (⋂U∈B. f-(U)) ⊆ f-(⋂B)
  proof
    fix x assume A4: x ∈ (⋂U∈B. f-(U))
    from A3 have ∀U∈B. f-(U) ⊆ X using func1_1_L6A by simp
    with A4 have ∀U∈B. x∈X by auto
    with A2 have x∈X by auto
    with A3 have ∃!y. ⟨ x,y⟩ ∈ f using Pi_iff_old by simp
    with A2 A4 show x ∈ f-(⋂B) using vimage_iff by blast
  qed
qed
```

The inverse image of a set does not change when we intersect the set with the image of the domain.

```
lemma inv_im_inter_im: assumes f:X→Y
  shows f-(A ∩ f(X)) = f-(A)
  using assms invim_inter_inter_invim inv_im_dom func1_1_L6A
  by blast
```

If the inverse image of a set is not empty, then the set is not empty. Proof by contradiction.

```
lemma func1_1_L13: assumes A1:f-(A) ≠ ∅ shows A≠∅
  using assms by auto
```

If the image of a set is not empty, then the set is not empty. Proof by contradiction.

```
lemma func1_1_L13A: assumes A1: f(A)≠∅ shows A≠∅
  using assms by auto
```

What is the inverse image of a singleton?

```
lemma func1_1_L14: assumes f:X→Y
  shows f-({y}) = {x∈X. f(x) = y}
```

```
  using assms func1_1_L6A vimage_singleton_iff apply_iff by auto
```

A lemma that can be used instead `fun_extension_iff` to show that two functions are equal

```
lemma func_eq:
  assumes f: X→Y  g: X→Zand  ∀x∈X. f(x) = g(x)
  shows f = g using assms fun_extension_iff by simp
```

An alternative syntax for defining a function: instead of writing $\{\langle x, p(x)\rangle . x \in X\}$ we can write $\lambda x \in X . p(x)$.

```
lemma lambda_fun_alt: shows {⟨x,p(x)⟩. x∈X} = (λx∈X. p(x))
proof -
  let L = {⟨x,p(x)⟩. x∈X}
  let R = λx∈X. p(x)
  have L:X→range(L) and R:X→range(L)
    using lam_is_fun_range range_fun lam_funtype by simp_all
  moreover have ∀x∈X. L(x) = R(x) using ZF_fun_from_tot_val1 beta by
simp
  ultimately show L = R using func_eq by blast
qed
```

If a function is equal to an expression $b(x)$ on X, then it has to be of the form $\{\langle x, b(x)\rangle | x \in X\}$.

```
lemma func_eq_set_of_pairs: assumes f:X→Y ∀x∈X. f(x) = b(x)
  shows f = {⟨x, b(x)⟩. x ∈ X}
proof -
  from assms(1) have f = {⟨x, f(x)⟩. x ∈ X} using fun_is_set_of_pairs
    by simp
  with assms(2) show thesis by simp
qed
```

Function defined on a singleton is a single pair.

```
lemma func_singleton_pair: assumes A1: f : {a}→X
  shows f = {⟨a, f(a)⟩}
proof -
  let g = {⟨a, f(a)⟩}
  note A1
  moreover have g : {a} → {f(a)} using singleton_fun by simp
  moreover have ∀x ∈ {a}. f(x) = g(x) using singleton_apply
    by simp
  ultimately show f = g by (rule func_eq)
qed
```

A single pair is a function on a singleton. This is similar to `singleton_fun` from standard Isabelle/ZF.

```
lemma pair_func_singleton: assumes A1: y ∈ Y
  shows {⟨x,y⟩} : {x} → Y
proof -
```

```
  have {⟨x,y⟩} : {x} → {y} using singleton_fun by simp
  moreover from A1 have {y} ⊆ Y by simp
  ultimately show {⟨x,y⟩} : {x} → Y
    by (rule func1_1_L1B)
qed
```

The value of a pair on the first element is the second one.

```
lemma pair_val: shows {⟨x,y⟩}(x) = y
  using singleton_fun apply_equality by simp
```

A more familiar definition of inverse image.

```
lemma func1_1_L15: assumes A1: f:X→Y
  shows f-(A) = {x∈X. f(x) ∈ A}
proof -
  have f-(A) = (⋃y∈A . f-{y})
    by (rule vimage_eq_UN)
  with A1 show thesis using func1_1_L14 by auto
qed
```

For symmetric functions inverse images are symmetric.

```
lemma symm_vimage_symm:
  assumes f:X×X→Y and ∀x∈X. ∀y∈X. f⟨x,y⟩ = f⟨y,x⟩
  shows f-(A) = converse(f-(A))
  using assms func1_1_L15 by auto
```

A more familiar definition of image.

```
lemma func_imagedef: assumes A1: f:X→Y and A2: A⊆X
  shows f(A) = {f(x). x ∈ A}
proof
  from A1 show f(A) ⊆ {f(x). x ∈ A}
    using image_iff apply_iff by auto
  show {f(x). x ∈ A} ⊆ f(A)
  proof
    fix y assume y ∈ {f(x). x ∈ A}
    then obtain x where x∈A and  y = f(x)
      by auto
    with A1 A2 have ⟨x,y⟩ ∈ f using apply_iff by force
    with A1 A2 ‹x∈A› show y ∈ f(A) using image_iff by auto
  qed
qed
```

If all elements of a nonempty set map to the same element of the codomain, then the image of this set is a singleton.

```
lemma image_constant_singleton:
  assumes f:X→Y A⊆X A≠∅ ∀x∈A. f(x) = c
  shows f(A) = {c}
  using assms func_imagedef by auto
```

A technical lemma about graphs of functions: if we have two disjoint sets A and B then the cartesian product of the inverse image of A and B is disjoint with (the graph of) f.

```
lemma vimage_prod_dis_graph: assumes f:X→Y A∩B = ∅
  shows f-(A)×B ∩ f = ∅
proof -
  { assume f-(A)×B ∩ f ≠ ∅
    then obtain p where p ∈ f-(A)×B and p∈f by blast
    from assms(1) ‹p∈f› have p ∈ {⟨x, f(x)⟩. x ∈ X}
      using fun_is_set_of_pairs by simp
    then obtain x where p = ⟨x, f(x)⟩ by blast
    with assms ‹p ∈ f-(A)×B› have False using func1_1_L15 by auto
  } thus thesis by auto
qed
```

For two functions with the same domain X and the codomain Y, Z resp., we can define a third one that maps X to the cartesian product of Y and Z.

```
lemma prod_fun_val:
  assumes {⟨x,p(x)⟩. x∈X}: X→Y {⟨x,q(x)⟩. x∈X}: X→Z
  defines h ≡ {⟨x,⟨p(x),q(x)⟩⟩. x∈X}
  shows h:X→Y×Z and ∀x∈X. h(x) = ⟨p(x),q(x)⟩
proof -
  from assms(1,2) have ∀x∈X. ⟨p(x),q(x)⟩ ∈ Y×Z
    using ZF_fun_from_tot_val(2) by auto
  with assms(3) show h:X→Y×Z using ZF_fun_from_total by simp
  with assms(3) show ∀x∈X. h(x) = ⟨p(x),q(x)⟩ using ZF_fun_from_tot_val0
    by simp
qed
```

Suppose we have two functions $f : X \to Y$ and $g : X \to Z$ and the third one is defined as $h : X \to Y \times Z$, $x \mapsto \langle f(x), g(x)\rangle$. Given two sets U, V we have $h^{-1}(U \times V) = (f^{-1}(U)) \cap (g^{-1}(V))$. We also show that the set where the function f, g are equal is the same as $h^{-1}(\{\langle y, y\rangle : y \in X\}$. It is a bit surprising that we get the last identity without the assumption that $Y = Z$.

```
lemma vimage_prod:
  assumes f:X→Y g:X→Z
  defines h ≡ {⟨x,⟨f(x),g(x)⟩⟩. x∈X}
  shows
    h:X→Y×Z
    ∀x∈X. h(x) = ⟨f(x),g(x)⟩
    h-(U×V) = f-(U) ∩ g-(V)
    {x∈X. f(x) = g(x)} = h-({⟨y,y⟩. y∈Y})
proof -
  from assms show h:X→Y×Z using apply_funtype ZF_fun_from_total
    by simp
  with assms(3) show I: ∀x∈X. h(x) = ⟨f(x),g(x)⟩
    using ZF_fun_from_tot_val by simp
  with assms(1,2) ‹h:X→Y×Z› show h-(U×V) = f-(U) ∩ g-(V)
```

```
    using func1_1_L15 by auto
  from assms(1) I ‹h:X→Y×Z› show {x∈X. f(x) = g(x)} = h-({⟨y,y⟩. y∈Y})
    using apply_funtype func1_1_L15 by auto
qed
```

The image of a set contained in domain under identity is the same set.

```
lemma image_id_same: assumes A⊆X shows id(X)(A) = A
  using assms id_type id_conv by auto
```

The inverse image of a set contained in domain under identity is the same set.

```
lemma vimage_id_same: assumes A⊆X shows id(X)-(A) = A
  using assms id_type id_conv by auto
```

What is the image of a singleton?

```
lemma singleton_image:
  assumes f∈X→Y and x∈X
  shows f{x} = {f(x)}
  using assms func_imagedef by auto
```

If an element of the domain of a function belongs to a set, then its value belongs to the image of that set.

```
lemma func1_1_L15D: assumes f:X→Y  x∈A  A⊆X
  shows f(x) ∈ f(A)
  using assms func_imagedef by auto
```

Range is the image of the domain. Isabelle/ZF defines `range(f)` as `domain(converse(f))`, and that's why we have something to prove here.

```
lemma range_image_domain:
  assumes A1: f:X→Y shows f(X) = range(f)
proof
  show f(X) ⊆ range(f) using image_def by auto
  { fix y assume y ∈ range(f)
    then obtain x where ⟨y,x⟩ ∈ converse(f) by auto
    with A1 have x∈X using func1_1_L5 by blast
    with A1 have f(x) ∈ f(X) using func_imagedef
      by auto
    with A1  ‹⟨y,x⟩ ∈ converse(f)› have y ∈ f(X)
      using apply_equality by auto
  } then show range(f) ⊆ f(X) by auto
qed
```

The difference of images is contained in the image of difference.

```
lemma diff_image_diff: assumes A1: f: X→Y and A2: A⊆X
  shows f(X)\f(A) ⊆ f(X\A)
proof
  fix y assume y ∈ f(X) \ f(A)
```

```
  hence y ∈ f(X) and I: y ∉ f(A) by auto
  with A1 obtain x where x∈X and II: y = f(x)
    using func_imagedef by auto
  with A1 A2 I have x∉A
    using func1_1_L15D by auto
  with ‹x∈X› have x ∈ X\A X\A ⊆ X by auto
  with A1 II show y ∈ f(X\A)
    using func1_1_L15D by simp
qed
```

The image of an intersection is contained in the intersection of the images.

```
lemma image_of_Inter: assumes  A1: f:X→Y and
  A2: I≠∅ and A3: ∀i∈I. P(i) ⊆ X
  shows f(⋂i∈I. P(i)) ⊆ ( ⋂i∈I. f(P(i)) )
proof
  fix y assume A4: y ∈ f(⋂i∈I. P(i))
  from A1 A3 have f(⋂i∈I. P(i)) = {f(x). x ∈ ( ⋂i∈I. P(i) )}
    using inter_subsets_subset func_imagedef by simp
  with A4 obtain x where x ∈ ( ⋂i∈I. P(i) ) and y = f(x)
    by auto
  with A1 A2 A3 show y ∈ ( ⋂i∈I. f(P(i)) ) using func_imagedef
    by auto
qed
```

The image of union is the union of images.

```
lemma image_of_Union: assumes A1: f:X→Y and A2: ∀A∈M. A⊆X
  shows f(⋃M) = ⋃{f(A). A∈M}
proof
  from A2 have ⋃M ⊆ X by auto
  { fix y assume y ∈ f(⋃M)
    with A1 ‹⋃M ⊆ X› obtain x where x∈⋃M and I: y = f(x)
      using func_imagedef by auto
    then obtain A where A∈M and x∈A by auto
    with assms I have y ∈ ⋃{f(A). A∈M} using func_imagedef by auto
  } thus f(⋃M) ⊆ ⋃{f(A). A∈M} by auto
  { fix y assume y ∈ ⋃{f(A). A∈M}
    then obtain A where A∈M and y ∈ f(A) by auto
    with assms ‹⋃M ⊆ X› have y ∈ f(⋃M) using func_imagedef by auto
  } thus ⋃{f(A). A∈M} ⊆ f(⋃M) by auto
qed
```

If the domain of a function is nonempty, then the codomain is as well.

```
lemma codomain_nonempty: assumes f:X→Y X≠∅ shows Y≠∅
  using assms apply_funtype by blast
```

The image of a nonempty subset of domain is nonempty.

```
lemma func1_1_L15A:
  assumes A1: f: X→Y and A2: A⊆X and A3: A≠∅
```

```
  shows f(A) ≠ ∅
proof -
  from A3 obtain x where x∈A by auto
  with A1 A2 have f(x) ∈ f(A)
    using func_imagedef by auto
  then show f(A) ≠ ∅ by auto
qed
```

The next lemma allows to prove statements about the values in the domain of a function given a statement about values in the range.

```
lemma func1_1_L15B:
  assumes f:X→Y and A⊆X and ∀y∈f(A). P(y)
  shows ∀x∈A. P(f(x))
  using assms func_imagedef by simp
```

An image of an image is the image of a composition.

```
lemma func1_1_L15C: assumes  A1: f:X→Y and A2: g:Y→Z
  and A3: A⊆X
  shows
  g(f(A)) =  {g(f(x)). x∈A}
  g(f(A)) = (g O f)(A)
proof -
  from A1 A3 have {f(x). x∈A} ⊆ Y
    using apply_funtype by auto
  with A2 have g{f(x). x∈A} = {g(f(x)). x∈A}
    using func_imagedef by auto
  with A1 A3 show I: g(f(A)) =  {g(f(x)). x∈A}
    using func_imagedef by simp
  from A1 A3 have ∀x∈A. (g O f)(x) = g(f(x))
    using comp_fun_apply by auto
  with I have g(f(A)) = {(g O f)(x). x∈A}
    by simp
  moreover from A1 A2 A3 have (g O f)(A) = {(g O f)(x). x∈A}
    using comp_fun func_imagedef by blast
  ultimately show g(f(A)) = (g O f)(A)
    by simp
qed
```

What is the image of a set defined by a meta-function?

```
lemma func1_1_L17:
  assumes A1: f ∈ X→Y and A2: ∀x∈A. b(x) ∈ X
  shows f({b(x). x∈A}) = {f(b(x)). x∈A}
proof -
  from A2 have {b(x). x∈A} ⊆ X by auto
  with A1 show thesis using func_imagedef by auto
qed
```

What are the values of composition of three functions?

```
lemma func1_1_L18: assumes A1: f:A→B  g:B→C  h:C→D
```

```
  and A2: x∈A
  shows
  (h O g O f)(x) ∈ D
  (h O g O f)(x) = h(g(f(x)))
proof -
  from A1 have (h O g O f) : A→D
    using comp_fun by blast
  with A2 show (h O g O f)(x) ∈ D using apply_funtype
    by simp
  from A1 A2 have (h O g O f)(x) = h( (g O f)(x))
    using comp_fun comp_fun_apply by blast
  with A1 A2 show (h O g O f)(x) = h(g(f(x)))
    using comp_fun_apply by simp
qed
```

A composition of functions is a function. This is a slight generalization of standard Isabelle's `comp_fun`.

```
lemma comp_fun_subset:
  assumes A1: g:A→B  and A2: f:C→D and A3: B ⊆ C
  shows f O g : A → D
proof -
  from A1 A3 have g:A→C by (rule func1_1_L1B)
  with A2 show f O g : A → D using comp_fun by simp
qed
```

This lemma supersedes the lemma `comp_eq_id_iff` in Isabelle/ZF. Contributed by Victor Porton.

```
lemma comp_eq_id_iff1: assumes A1: g: B→A and A2: f: A→C
  shows (∀y∈B. f(g(y)) = y) ⟷ f O g = id(B)
proof -
  from assms have f O g: B→C and id(B): B→B
    using comp_fun id_type by auto
  then have (∀y∈B. (f O g)y = id(B)(y)) ⟷ f O g = id(B)
    by (rule fun_extension_iff)
  moreover from A1 have
    ∀y∈B. (f O g)y = f(gy) and ∀y∈B. id(B)(y) = y
    by auto
  ultimately show (∀y∈B. f(gy) = y) ⟷ f O g = id(B) by simp
qed
```

A lemma about a value of a function that is a union of some collection of functions.

```
lemma fun_Union_apply: assumes A1: ⋃F : X→Y and
  A2: f∈F and A3: f:A→B and A4: x∈A
  shows (⋃F)(x) = f(x)
proof -
  from A3 A4 have ⟨x, f(x)⟩ ∈ f using apply_Pair
    by simp
```

```
  with A2 have ⟨x, f(x)⟩ ∈ ⋃F by auto
  with A1 show (⋃F)(x) = f(x) using apply_equality
    by simp
qed
```

9.2 Dependent function space

The standard Isabelle/ZF `ZF_Base` theory defines a general notion of a dependent function space `Pi(X,N)`, where X is any set and N is a collection of sets indexed by X. We rarely use that notion in IsarMathLib. The facts shown in this section provide information on how to interpret the dependent function space notion in terms of a regular space of functions defined on X with values in $Y = \bigcup_{x\in X} N(x)$.

The `Pi_iff_old` lemma from the standard Isabelle/ZF `func` theory shows that if f is a member of the dependent function space `Pi(X,N)` and $x \in X$ then there exist exactly one y such that $\langle x, y\rangle \in f$. The next lemma shows that we can slightly strengthen this assertion and claim that there exist exactly one $y \in N(x)$ such that $\langle x, y\rangle \in f$. Consequently, there exists exactly one $y \in \bigcup_{t\in X} N(t)$ such that $\langle x, y\rangle \in f$.

```
lemma pi_then: assumes f∈Pi(X,N) x∈X
  shows ∃!y. y∈N(x) ∧ ⟨x,y⟩ ∈ f and ∃!y. y∈(⋃t∈X. N(t)) ∧ ⟨x,y⟩ ∈ f
proof -
  from assms have f(x)∈N(x) and ∃!y. ⟨x,y⟩ ∈ f
    using Pi_iff_old apply_type by simp_all
  with assms show ∃!y. y∈N(x) ∧ ⟨x,y⟩ ∈ f and ∃!y. y∈(⋃t∈X. N(t))
∧ ⟨x,y⟩ ∈ f
    using apply_equality by auto
qed
```

The next lemma demonstrates a way to understand the dependent function space `Pi(X,N)`: it is the space of functions that map X into $\bigcup_{t\in X} N(t)$ such that for all $x \in X$ we have $f(x) \in N(x)$.

```
theorem pi_fun_space: shows Pi(X,N) = {f∈X→(⋃x∈X. N(x)). ∀x∈X. f(x)∈N(x)}
proof
  let Y = (⋃x∈X. N(x))
  let R = {f∈X→Y. ∀x∈X. f(x)∈N(x)}
  { fix f assume f∈Pi(X,N)
    then have f⊆Sigma(X,N) using Pi_iff by auto
    hence f ⊆ X×(⋃x∈X. N(x)) by auto
    with ‹f∈Pi(X,N)› have f: X→Y and ∀x∈X. f(x)∈N(x)
      using pi_then(2) func1_1_L11 apply_type by simp_all
  } thus Pi(X,N) ⊆ R by auto
  { fix f assume f∈R
    then have f:X→Y and ∀x∈X. f(x)∈N(x)
      by auto
    { fix p assume p∈f
```

```
      let x = fst(p)
      let y = snd(p)
      from ‹f:X→Y› have f ⊆ X×Y using fun_subset_prod by simp
      with ‹p∈f› have p = ⟨x,y⟩ and ⟨x,y⟩ ∈ f by auto
      from ‹f:X→Y› ‹⟨x,y⟩ ∈ f› have x∈X and y=f(x)
        using pair_fun_member by simp_all
      with ‹∀x∈X. f(x)∈N(x)› have ⟨x,y⟩ ∈ Sigma(X,N) using SigmaI by
simp
      with ‹p = ⟨x,y⟩› have p∈Sigma(X,N) by simp
    } hence f⊆Sigma(X,N) by auto
    from ‹f:X→Y› have function(f) and X = domain(f)
      using func1_1_L1 fun_is_fun by simp_all
    with ‹f⊆Sigma(X,N)› have f∈Pi(X,N) using Pi_iff by simp
  } thus R ⊆ Pi(X,N) by auto
qed
```

9.3 Functions restricted to a set

Standard Isabelle/ZF defines the notion `restrict(f,A)` of to mean a function (or relation) f restricted to a set. This means that if f is a function defined on X and A is a subset of X then `restrict(f,A)` is a function whith the same values as f, but whose domain is A.

What is the inverse image of a set under a restricted fuction?

```
lemma func1_2_L1: assumes A1: f:X→Y and A2: B⊆X
  shows restrict(f,B)-(A) = f-(A) ∩ B
proof -
  let g = restrict(f,B)
  from A1 A2 have g:B→Y
    using restrict_type2 by simp
  with A2 A1 show g-(A) = f-(A) ∩ B
    using func1_1_L15 restrict_if by auto
qed
```

A criterion for when one function is a restriction of another. The lemma below provides a result useful in the actual proof of the criterion and applications.

```
lemma func1_2_L2:
  assumes A1: f:X→Y and A2: g ∈ A→Z
  and A3: A⊆X and A4: f ∩ A×Z = g
  shows ∀x∈A. g(x) = f(x)
proof
  fix x assume x∈A
  with A2 have ⟨x,g(x)⟩ ∈ g using apply_Pair by simp
  with A4 A1 show g(x) = f(x)  using apply_iff by auto
qed
```

Here is the actual criterion.

```
lemma func1_2_L3:
  assumes A1: f:X→Y and A2: g:A→Z
  and A3: A⊆X and A4: f ∩ A×Z = g
  shows g = restrict(f,A)
proof
  from A4 show g ⊆ restrict(f, A) using restrict_iff by auto
  show restrict(f, A) ⊆ g
  proof
    fix z assume A5:z ∈ restrict(f,A)
    then obtain x y where D1:z∈f ∧ x∈A  ∧ z = ⟨x,y⟩
      using restrict_iff by auto
    with A1 have y = f(x) using apply_iff by auto
    with A1 A2 A3 A4 D1 have y = g(x) using func1_2_L2 by simp
    with A2 D1 show z∈g using apply_Pair by simp
  qed
qed
```

Which function space a restricted function belongs to?

```
lemma func1_2_L4:
  assumes A1: f:X→Y and A2: A⊆X and A3: ∀x∈A. f(x) ∈ Z
  shows restrict(f,A) : A→Z
proof -
  let g = restrict(f,A)
  from A1 A2 have g : A→Y
    using restrict_type2 by simp
  moreover {
    fix x assume x∈A
    with A1 A3 have g(x) ∈ Z using restrict by simp}
  ultimately show thesis by (rule Pi_type)
qed
```

A simpler case of `func1_2_L4`, where the range of the original and restricted function are the same.

```
corollary restrict_fun: assumes A1: f:X→Y and A2: A⊆X
  shows restrict(f,A) : A → Y
proof -
  from assms have ∀x∈A. f(x) ∈ Y using apply_funtype
    by auto
  with assms show thesis using func1_2_L4 by simp
qed
```

A function restricted to its domain is itself.

```
lemma restrict_domain: assumes f:X→Y
  shows restrict(f,X) = f
proof -
  have ∀x∈X. restrict(f,X)(x) = f(x) using restrict by simp
  with assms show thesis using func_eq restrict_fun by blast
qed
```

Suppose a function $f : X \to Y$ is defined by an expression q, i.e. $f = \{\langle x, y\rangle : x \in X\}$. Then a function that is defined by the same expression, but on a smaller set is the same as the restriction of f to that smaller set.

```
lemma restrict_def_alt: assumes A⊆X
  shows restrict({⟨x,q(x)⟩. x∈X},A) = {⟨x,q(x)⟩. x∈A}
proof -
  let Y = {q(x). x∈X}
  let f = {⟨x,q(x)⟩. x∈X}
  have ∀x∈X. q(x)∈Y by blast
  with assms have f:X→Y using ZF_fun_from_total by simp
  with assms have restrict(f,A):A→Y using restrict_fun by simp
  moreover
  from assms have ∀x∈A. q(x)∈Y by blast
  then have {⟨x,q(x)⟩. x∈A}:A→Y using ZF_fun_from_total by simp
  moreover from assms have
    ∀x∈A. restrict(f,A)(x) = {⟨x,q(x)⟩. x∈A}(x)
    using restrict ZF_fun_from_tot_val1 by auto
  ultimately show thesis by (rule func_eq)
qed
```

A composition of two functions is the same as composition with a restriction.

```
lemma comp_restrict:
  assumes A1: f : A→B and A2: g : X → C and A3: B⊆X
  shows g O f = restrict(g,B) O f
proof -
  from assms have g O f : A → C using comp_fun_subset
    by simp
  moreover from assms have restrict(g,B) O f : A → C
    using restrict_fun comp_fun by simp
  moreover from A1 have
    ∀x∈A. (g O f)(x) = (restrict(g,B) O f)(x)
    using comp_fun_apply apply_funtype restrict
    by simp
  ultimately show g O f = restrict(g,B) O f
    by (rule func_eq)
qed
```

A way to look at restriction. Contributed by Victor Porton.

```
lemma right_comp_id_any: shows r O id(C) = restrict(r,C)
  unfolding restrict_def by auto
```

9.4 Constant functions

Constant functions are trivial, but still we need to prove some properties to shorten proofs.

We define constant$(= c)$ functions on a set X in a natural way as ConstantFunction(X, c).

```
definition
```

```
  ConstantFunction(X,c) ≡ X×{c}
```

Constant function is a function (i.e. belongs to a function space).

```
lemma func1_3_L1:
  assumes A1: c∈Y shows ConstantFunction(X,c) : X→Y
proof -
   from A1 have X×{c} = {⟨x,y⟩ ∈ X×Y. c = y}
     by auto
   with A1 show thesis using func1_1_L11A ConstantFunction_def
     by simp
qed
```

Constant function is equal to the constant on its domain.

```
lemma func1_3_L2: assumes A1: x∈X
  shows ConstantFunction(X,c)(x) = c
proof -
  have ConstantFunction(X,c) ∈ X→{c}
    using func1_3_L1 by simp
  moreover from A1 have ⟨x,c⟩ ∈ ConstantFunction(X,c)
    using ConstantFunction_def by simp
  ultimately show thesis using apply_iff by simp
qed
```

Another way of looking at the constant function - it's a set of pairs $\langle x, c\rangle$ as x ranges over X.

```
lemma const_fun_def_alt: shows ConstantFunction(X,c) = {⟨x,c⟩. x∈X}
  unfolding ConstantFunction_def by auto
```

Yet another definition of a constant function: it's a cartesian product of its domain and the singleton of its value.

```
lemma const_fun_def_alt1: shows ConstantFunction(X,c) = X×{c}
  using const_fun_def_alt by auto
```

If $c \in A$ then the inverse image of A by the constant function $x \mapsto c$ is the whole domain.

```
lemma const_vimage_domain: assumes c∈A
  shows ConstantFunction(X,c)-(A) = X
proof -
  let C = ConstantFunction(X,c)
  have C-(A) = {x∈X. C(x) ∈ A} using func1_3_L1 func1_1_L15
    by blast
  with assms show thesis using func1_3_L2 by simp
qed
```

If c is not an element of A then the inverse image of A by the constant function $x \mapsto c$ is empty.

```
lemma const_vimage_empty: assumes c∉A
```

```
  shows ConstantFunction(X,c)-(A) = ∅
proof -
  let C = ConstantFunction(X,c)
  have C-(A) = {x∈X. C(x) ∈ A} using func1_3_L1 func1_1_L15
    by blast
  with assms show thesis using func1_3_L2 by simp
qed
```

9.5 Injections, surjections, bijections etc.

In this section we prove the properties of the spaces of injections, surjections and bijections that we can't find in the standard Isabelle's `Perm.thy`.

For injections the image a difference of two sets is the difference of images

```
lemma inj_image_dif:
  assumes A1: f ∈ inj(A,B) and A2: C ⊆ A
  shows f(A\C) = f(A)\f(C)
proof
  show f(A\C) ⊆ f(A)\f(C)
  proof
    fix y assume A3: y ∈ f(A\C)
    from A1 have f:A→B using inj_def by simp
    moreover have A\C ⊆ A by auto
    ultimately have f(A\C) = {f(x). x ∈ A\C}
      using func_imagedef by simp
    with A3 obtain x where I: f(x) = y and x ∈ A\C
      by auto
    hence x∈A by auto
    with ‹f:A→B› I have y ∈ f(A)
      using func_imagedef by auto
    moreover have y ∉  f(C)
    proof -
      { assume y ∈ f(C)
 with A2 ‹f:A→B› obtain x0
   where II: f(x0) = y and x0 ∈ C
   using func_imagedef by auto
 with A1 A2 I ‹x∈A› have
   f ∈ inj(A,B) f(x) = f(x0)  x∈A x0 ∈ A
   by auto
 then have x = x0 by (rule inj_apply_equality)
 with ‹x ∈ A\C› ‹x0 ∈ C› have False by simp
      } thus thesis by auto
    qed
    ultimately show y ∈ f(A)\f(C) by simp
  qed
  from A1 A2 show f(A)\f(C) ⊆ f(A\C)
    using inj_def diff_image_diff by auto
qed
```

For injections the image of intersection is the intersection of images.

```
lemma inj_image_inter: assumes A1: f ∈ inj(X,Y) and A2: A⊆X B⊆X
  shows f(A∩B) = f(A) ∩ f(B)
proof
  show f(A∩B) ⊆ f(A) ∩ f(B) using image_Int_subset by simp
  { from A1 have f:X→Y using inj_def by simp
    fix y assume y ∈ f(A) ∩ f(B)
    then have y ∈ f(A) and  y ∈ f(B) by auto
    with A2 ‹f:X→Y› obtain x_A x_B where
    x_A ∈ A x_B ∈ B and I: y = f(x_A)  y = f(x_B)
      using func_imagedef by auto
    with A2 have x_A ∈ X x_B ∈ X and  f(x_A) =  f(x_B) by auto
    with A1 have x_A = x_B using inj_def by auto
    with ‹x_A ∈ A› ‹x_B ∈ B› have f(x_A) ∈ {f(x). x ∈ A∩B} by auto
    moreover from A2 ‹f:X→Y› have f(A∩B) = {f(x). x ∈ A∩B}
      using func_imagedef by blast
    ultimately have f(x_A) ∈ f(A∩B) by simp
    with I have y ∈ f(A∩B) by simp
  } thus f(A) ∩ f(B) ⊆ f(A ∩ B) by auto
qed
```

For surjection from A to B the image of the domain is B.

```
lemma surj_range_image_domain: assumes A1: f ∈ surj(A,B)
  shows f(A) = B
proof -
  from A1 have f(A) = range(f)
    using surj_def range_image_domain by auto
  with A1 show f(A) = B  using surj_range
    by simp
qed
```

Surjections are functions that map the domain onto the codomain.

```
lemma surj_def_alt: shows surj(X,Y) = {f∈X→Y. f(X) = Y}
proof
  show surj(X,Y) ⊆ {f∈X→Y. f(X) = Y}
    using surj_range_image_domain unfolding surj_def by auto
  show {f∈X→Y. f(X) = Y} ⊆ surj(X,Y)
    using range_image_domain fun_is_surj by auto
qed
```

Bijections are functions that preserve complements.

```
lemma bij_def_alt:
  shows bij(X,Y) = {f∈X→Y. ∀A∈Pow(X). f(X\A) = Y\f(A)}
proof
  let R = {f∈X→Y. ∀A∈Pow(X). f(X\A) = Y\f(A)}
  show bij(X,Y) ⊆ R
    using inj_image_dif surj_range_image_domain surj_is_fun
    unfolding bij_def by auto
```

```
  { fix f assume f∈R
    hence f:X→Y and I: ∀A∈Pow(X). f(X\A) = Y\f(A)
      by auto
    { fix x1 x2 assume x1∈X x2∈X f(x1) = f(x2)
      with ‹f:X→Y› have
        f{x1} = {f(x1)} f{x2} = {f(x2)} f{x1} = f{x2}
        using singleton_image by simp_all
      { assume x1≠x2
        from ‹f:X→Y› have f(X\{x1}) = {f(t). t∈X\{x1}}
          using func_imagedef by blast
        with I ‹x2∈X› ‹x1∈X› ‹x1≠x2› ‹f{x1} = f{x2}›
          have f(x2) ∈ Y\f{x2} by auto
        with ‹f{x2} = {f(x2)}› have False by auto
      } hence x1=x2 by auto
    } with ‹f:X→Y› have f∈inj(X,Y) unfolding inj_def
      by auto
    moreover
    from I have f(X\∅) = Y\f(∅) by blast
    with ‹f:X→Y› have f∈surj(X,Y) using surj_def_alt by simp
    ultimately have f ∈ bij(X,Y) unfolding bij_def by simp
  } thus R ⊆ bij(X,Y) by auto
qed
```

For injections the inverse image of an image is the same set.

```
lemma inj_vimage_image: assumes f ∈ inj(X,Y) and A⊆X
  shows f-(f(A)) = A
proof -
  have f-(f(A)) = (converse(f) O f)(A)
    using vimage_converse image_comp by simp
  with assms show thesis using left_comp_inverse image_id_same
    by simp
qed
```

For surjections the image of an inverse image is the same set.

```
lemma surj_image_vimage: assumes A1: f ∈ surj(X,Y) and A2: A⊆Y
  shows f(f-(A)) = A
proof -
  have f(f-(A)) = (f O converse(f))(A)
    using vimage_converse image_comp by simp
  with assms show thesis using right_comp_inverse image_id_same
    by simp
qed
```

A lemma about how a surjection maps collections of subsets in domain and range.

```
lemma surj_subsets: assumes A1: f ∈ surj(X,Y) and A2: B ⊆ Pow(Y)
  shows { f(U). U ∈ {f-(V). V∈B} } = B
proof
  { fix W assume W ∈ { f(U). U ∈ {f-(V). V∈B} }
```

```
    then obtain U where I: U ∈ {f-(V). V∈B} and II: W = f(U) by auto
    then obtain V where V∈B and U = f-(V) by auto
    with II have W = f(f-(V)) by simp
    moreover from assms ‹V∈B› have f ∈ surj(X,Y) and V⊆Y by auto
    ultimately have W=V using surj_image_vimage by simp
    with ‹V∈B› have W ∈ B by simp
  } thus { f(U). U ∈ {f-(V). V∈B} } ⊆ B by auto
  { fix W assume W∈B
    let U = f-(W)
    from ‹W∈B› have U ∈ {f-(V). V∈B} by auto
    moreover from A1 A2 ‹W∈B› have W = f(U) using surj_image_vimage
by auto
    ultimately have W ∈ { f(U). U ∈ {f-(V). V∈B} } by auto
  } thus B ⊆ { f(U). U ∈ {f-(V). V∈B} } by auto
qed
```

Restriction of an bijection to a set without a point is a a bijection.

```
lemma bij_restrict_rem:
  assumes A1: f ∈ bij(A,B) and A2: a∈A
  shows restrict(f, A\{a}) ∈ bij(A\{a}, B\{f(a)})
proof -
  let C = A\{a}
  from A1 have f ∈ inj(A,B)  C ⊆ A
    using bij_def by auto
  then have restrict(f,C) ∈ bij(C, f(C))
    using restrict_bij by simp
  moreover have f(C) =  B\{f(a)}
  proof -
    from A2 ‹f ∈ inj(A,B)› have f(C) = f(A)\f{a}
      using inj_image_dif by simp
    moreover from A1 have f(A) = B
      using bij_def surj_range_image_domain by auto
    moreover from A1 A2 have f{a} = {f(a)}
      using bij_is_fun singleton_image by blast
    ultimately show f(C) =  B\{f(a)} by simp
  qed
  ultimately show thesis by simp
qed
```

The domain of a bijection between X and Y is X.

```
lemma domain_of_bij:
  assumes A1: f ∈ bij(X,Y) shows domain(f) = X
proof -
  from A1 have f:X→Y using bij_is_fun by simp
  then show domain(f) = X using func1_1_L1 by simp
qed
```

The value of the inverse of an injection on a point of the image of a set belongs to that set.

```
lemma inj_inv_back_in_set:
  assumes A1: f ∈ inj(A,B) and A2: C⊆A and A3: y ∈ f(C)
  shows
  converse(f)(y) ∈ C
  f(converse(f)(y)) = y
proof -
  from A1 have I: f:A→B using inj_is_fun by simp
  with A2 A3 obtain x where II: x∈C   y = f(x)
    using func_imagedef by auto
  with A1 A2 show converse(f)(y) ∈ C using left_inverse
    by auto
  from A1 A2 I II show f(converse(f)(y)) = y
    using func1_1_L5A right_inverse by auto
qed
```

For a bijection between Y and X and a set $A \subseteq X$ an element $y \in Y$ is in the image $f(A)$ if and only if $f^{-1}(y)$ is an element of A. Note this is false with the weakened assumption that f is an injection, for example consider $f : \{0,1\} \to \mathbb{N}, f(n) = n+1$ and $y = 3$. Then $f^{-1} : \{1,2\} \to \{0,1\}$ and (since 3 is not in the domain of the inverse function) $f^{-1}(3) = \emptyset = 0 \in \{0,1\}$, but 3 is not in the image $f(\{0,1\})$.

```
lemma bij_val_image_vimage: assumes f ∈ bij(X,Y) A⊆X y∈Y
  shows y∈f(A) ⟷ converse(f)(y) ∈ A
proof
  assume y∈f(A)
  with assms(1,2) show converse(f)(y) ∈ A
    unfolding bij_def using inj_inv_back_in_set by blast
next
  assume converse(f)(y) ∈ A
  with assms(1,3) have y∈{f(x). x∈A} using right_inverse_bij
    by force
  with assms(1,2) show y∈f(A) using bij_is_fun func_imagedef
    by force
qed
```

For injections if a value at a point belongs to the image of a set, then the point belongs to the set.

```
lemma inj_point_of_image:
  assumes A1: f ∈ inj(A,B) and A2: C⊆A and
  A3: x∈A and A4: f(x) ∈ f(C)
  shows x ∈ C
proof -
  from A1 A2 A4 have converse(f)(f(x)) ∈ C
    using inj_inv_back_in_set by simp
  moreover from A1 A3 have converse(f)(f(x)) = x
    using left_inverse_eq by simp
  ultimately show x ∈ C by simp
qed
```

For injections the image of intersection is the intersection of images.

```
lemma inj_image_of_Inter: assumes A1: f ∈ inj(A,B) and
  A2: I≠∅ and A3: ∀i∈I. P(i) ⊆ A
  shows f(⋂i∈I. P(i)) = ( ⋂i∈I. f(P(i)) )
proof
  from A1 A2 A3 show f(⋂i∈I. P(i)) ⊆ ( ⋂i∈I. f(P(i)) )
    using inj_is_fun image_of_Inter by auto
  from A1 A3 have f:A→B  and ( ⋂i∈I. P(i) ) ⊆ A
    using inj_is_fun inter_subsets_subset by auto
  then have I: f(⋂i∈I. P(i)) = { f(x). x ∈ ( ⋂i∈I. P(i) ) }
    using func_imagedef by simp
  { fix y assume A4: y ∈ ( ⋂i∈I. f(P(i)) )
    let x = converse(f)(y)
    from A2 obtain i₀ where i₀ ∈ I by auto
    with A1 A4 have II: y ∈ range(f) using inj_is_fun func1_1_L6
      by auto
    with A1 have III: f(x) = y using right_inverse by simp
    from A1 II have IV: x ∈ A using inj_converse_fun apply_funtype
      by blast
    { fix i assume i∈I
      with A3 A4 III have P(i) ⊆ A and f(x) ∈  f(P(i))
 by auto
      with A1 IV have x ∈ P(i) using inj_point_of_image
 by blast
    } then have ∀i∈I. x ∈ P(i) by simp
    with A2 I have f(x) ∈ f( ⋂i∈I. P(i) )
      by auto
    with III have y ∈  f( ⋂i∈I. P(i) ) by simp
  } then show ( ⋂i∈I. f(P(i)) ) ⊆  f( ⋂i∈I. P(i) )
    by auto
qed
```

An injection is injective onto its range. Suggested by Victor Porton.

```
lemma inj_inj_range: assumes f ∈ inj(A,B)
  shows f ∈ inj(A,range(f))
  using assms inj_def range_of_fun by auto
```

An injection is a bijection on its range. Suggested by Victor Porton.

```
lemma inj_bij_range: assumes f ∈ inj(A,B)
  shows f ∈ bij(A,range(f))
proof -
  from assms have f ∈ surj(A,range(f)) using inj_def fun_is_surj
    by auto
  with assms show thesis using inj_inj_range bij_def by simp
qed
```

A lemma about extending a surjection by one point.

```
lemma surj_extend_point:
```

```
  assumes A1: f ∈ surj(X,Y) and A2: a∉X and
  A3: g = f ∪ {⟨a,b⟩}
  shows g ∈ surj(X∪{a},Y∪{b})
proof -
  from A1 A2 A3 have g : X∪{a} → Y∪{b}
    using surj_def func1_1_L11D by simp
  moreover have ∀y ∈ Y∪{b}. ∃x ∈ X∪{a}. y = g(x)
  proof
    fix y assume y ∈  Y ∪ {b}
    then have y ∈ Y ∨ y = b by auto
    moreover
    { assume y ∈ Y
      with A1 obtain x where x∈X and y = f(x)
 using surj_def by auto
      with A1 A2 A3 have x ∈  X∪{a} and y = g(x)
 using surj_def func1_1_L11D by auto
      then have ∃x ∈ X∪{a}. y = g(x) by auto }
    moreover
    { assume y = b
      with A1 A2 A3 have y = g(a)
 using surj_def func1_1_L11D by auto
      then have ∃x ∈ X∪{a}. y = g(x) by auto }
    ultimately show ∃x ∈ X∪{a}. y = g(x)
      by auto
  qed
  ultimately show g ∈ surj(X∪{a},Y∪{b})
    using surj_def by auto
qed
```

A lemma about extending an injection by one point. Essentially the same as standard Isabelle's `inj_extend`.

```
lemma inj_extend_point: assumes f ∈ inj(X,Y) a∉X b∉Y
  shows (f ∪ {⟨a,b⟩}) ∈ inj(X∪{a},Y∪{b})
proof -
  from assms have cons(⟨a,b⟩,f) ∈ inj(cons(a, X), cons(b, Y))
    using assms inj_extend by simp
  moreover have cons(⟨a,b⟩,f) = f ∪ {⟨a,b⟩} and
    cons(a, X) = X∪{a} and cons(b, Y) = Y∪{b}
    by auto
  ultimately show thesis by simp
qed
```

A lemma about extending a bijection by one point.

```
lemma bij_extend_point: assumes f ∈ bij(X,Y) a∉X b∉Y
  shows (f ∪ {⟨a,b⟩}) ∈ bij(X∪{a},Y∪{b})
  using assms surj_extend_point inj_extend_point bij_def
  by simp
```

A quite general form of the $a^{-1}b = 1$ implies $a = b$ law.

```
lemma comp_inv_id_eq:
  assumes A1: converse(b) O a = id(A) and
  A2: a ⊆ A×B b ∈ surj(A,B)
  shows a = b
proof -
  from A1 have (b O converse(b)) O a = b O id(A)
    using comp_assoc by simp
  with A2 have id(B) O a = b O id(A)
    using right_comp_inverse by simp
  moreover
  from A2 have a ⊆ A×B and b ⊆ A×B
    using surj_def fun_subset_prod
    by auto
  then have id(B) O a = a and b O id(A) = b
    using left_comp_id right_comp_id by auto
  ultimately show a = b by simp
qed
```

A special case of `comp_inv_id_eq` - the $a^{-1}b = 1$ implies $a = b$ law for bijections.

```
lemma comp_inv_id_eq_bij:
  assumes A1: a ∈ bij(A,B) b ∈ bij(A,B) and
  A2: converse(b) O a = id(A)
  shows a = b
proof -
  from A1 have  a ⊆ A×B and b ∈ surj(A,B)
    using bij_def surj_def fun_subset_prod
    by auto
  with A2 show a = b by (rule comp_inv_id_eq)
qed
```

Converse of a converse of a bijection is the same bijection. This is a special case of `converse_converse` from standard Isabelle's `equalities` theory where it is proved for relations.

```
lemma bij_converse_converse: assumes a ∈ bij(A,B)
  shows converse(converse(a)) = a
proof -
  from assms have a ⊆ A×B using bij_def surj_def fun_subset_prod by
simp
  then show thesis using converse_converse by simp
qed
```

If a composition of bijections is identity, then one is the inverse of the other.

```
lemma comp_id_conv: assumes A1: a ∈ bij(A,B) b ∈ bij(B,A) and
  A2: b O a = id(A)
  shows a = converse(b) and b = converse(a)
proof -
  from A1 have a ∈ bij(A,B) and converse(b) ∈ bij(A,B) using bij_converse_bij
```

```
    by auto
  moreover from assms have converse(converse(b)) O a = id(A)
    using bij_converse_converse by simp
  ultimately show a = converse(b) by (rule comp_inv_id_eq_bij)
  with assms show b = converse(a) using bij_converse_converse by simp
qed
```

A version of comp_id_conv with weaker assumptions.

```
lemma comp_conv_id: assumes A1: a ∈ bij(A,B) and A2: b:B→A and
  A3: ∀x∈A. b(a(x)) = x
  shows b ∈ bij(B,A) and  a = converse(b) and b = converse(a)
proof -
  have b ∈ surj(B,A)
  proof -
    have ∀x∈A. ∃y∈B. b(y) = x
    proof -
      { fix x assume x∈A
        let y = a(x)
        from A1 A3 ‹x∈A› have y∈B and b(y) = x
          using bij_def inj_def apply_funtype by auto
        hence ∃y∈B. b(y) = x by auto
      } thus thesis by simp
    qed
    with A2 show b ∈ surj(B,A) using surj_def by simp
  qed
  moreover have b ∈ inj(B,A)
  proof -
    have ∀w∈B.∀y∈B. b(w) = b(y) ⟶ w=y
    proof -
      { fix w y assume w∈B  y∈B and I: b(w) = b(y)
        from A1 have a ∈ surj(A,B) unfolding bij_def by simp
        with ‹w∈B› obtain x_w where x_w ∈ A and II: a(x_w) = w
          using surj_def by auto
        with I have b(a(x_w)) = b(y) by simp
        moreover from ‹a ∈ surj(A,B)› ‹y∈B› obtain x_y where
          x_y ∈ A and III: a(x_y) = y
          using surj_def by auto
        moreover from A3 ‹x_w ∈ A›  ‹x_y ∈ A› have b(a(x_w)) = x_w and
b(a(x_y)) = x_y
          by auto
        ultimately have x_w = x_y by simp
        with II III have w=y by simp
      } thus thesis by auto
    qed
    with A2 show b ∈ inj(B,A) using inj_def by auto
  qed
  ultimately show b ∈ bij(B,A) using bij_def by simp
  from assms have b O a = id(A) using bij_def inj_def comp_eq_id_iff1
by auto
```

```
  with A1 ‹b ∈ bij(B,A)› show a = converse(b) and b = converse(a)
    using comp_id_conv by auto
qed
```

For a surjection the union if images of singletons is the whole range.

```
lemma surj_singleton_image: assumes A1: f ∈ surj(X,Y)
  shows (⋃x∈X. {f(x)}) = Y
proof
  from A1 show (⋃x∈X. {f(x)}) ⊆ Y
    using surj_def apply_funtype by auto
next
  { fix y assume y ∈ Y
    with A1 have y ∈ (⋃x∈X. {f(x)})
      using surj_def by auto
  } then show  Y ⊆ (⋃x∈X. {f(x)}) by auto
qed
```

9.6 Functions of two variables

In this section we consider functions whose domain is a cartesian product of two sets. Such functions are called functions of two variables (although really in ZF all functions admit only one argument). For every function of two variables we can define families of functions of one variable by fixing the other variable. This section establishes basic definitions and results for this concept.

We can create functions of two variables by combining functions of one variable.

```
lemma cart_prod_fun: assumes f₁:X₁→Y₁  f₂:X₂→Y₂ and
  g = {⟨p,⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ X₁×X₂}
  shows g: X₁×X₂ → Y₁×Y₂ using assms apply_funtype  ZF_fun_from_total
by simp
```

A reformulation of `cart_prod_fun` above in a sligtly different notation.

```
lemma prod_fun:
  assumes f:X₁→X₂  g:X₃→X₄
  shows {⟨⟨x,y⟩,⟨f(x),g(y)⟩⟩. ⟨x,y⟩∈X₁×X₃}:X₁×X₃→X₂×X₄
proof -
  have {⟨⟨x,y⟩,⟨f(x),g(y)⟩⟩. ⟨x,y⟩∈X₁×X₃} = {⟨p,⟨f(fst(p)),g(snd(p))⟩⟩.
p ∈ X₁×X₃}
    by auto
  with assms show thesis using cart_prod_fun by simp
qed
```

Product of two surjections is a surjection.

```
theorem prod_functions_surj:
  assumes f∈surj(A,B) g∈surj(C,D)
```

```
  shows {⟨⟨a₁,a₂⟩,⟨f(a₁),g(a₂)⟩⟩. ⟨a₁,a₂⟩∈A×C} ∈ surj(A×C,B×D)
proof -
  let h = {⟨⟨x, y⟩, f(x), g(y)⟩ . ⟨x,y⟩ ∈ A×C}
  from assms have fun: f:A→B g:C→D unfolding surj_def by auto
  then have pfun: h : A × C → B × D using prod_fun by auto
  {
    fix b assume b∈B×D
    then obtain b₁ b₂ where b=⟨b₁,b₂⟩ b₁∈B b₂∈D by auto
    with assms obtain a₁ a₂ where f(a₁)=b₁ g(a₂)=b₂ a₁∈A a₂∈C
      unfolding surj_def by blast
    hence ⟨⟨a₁,a₂⟩,⟨b₁,b₂⟩⟩ ∈ h by auto
    with pfun have h⟨a₁,a₂⟩=⟨b₁,b₂⟩ using apply_equality by auto
    with ‹b=⟨b₁,b₂⟩› ‹a₁∈A› ‹a₂∈C› have ∃a∈A×C. h(a)=b
      by auto
  } hence ∀b∈B×D. ∃a∈A×C. h(a) = b by auto
  with pfun show thesis unfolding surj_def by auto
qed
```

For a function of two variables created from functions of one variable as in `cart_prod_fun` above, the inverse image of a cartesian product of sets is the cartesian product of inverse images.

```
lemma cart_prod_fun_vimage: assumes f₁:X₁→Y₁  f₂:X₂→Y₂ and
  g = {⟨p,⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ X₁×X₂}
  shows g-(A₁×A₂) = f₁-(A₁) × f₂-(A₂)
proof -
  from assms have g: X₁×X₂ → Y₁×Y₂ using cart_prod_fun
    by simp
  then have g-(A₁×A₂) = {p ∈ X₁×X₂. g(p) ∈ A₁×A₂} using func1_1_L15

    by simp
  with assms ‹g: X₁×X₂ → Y₁×Y₂› show g-(A₁×A₂) = f₁-(A₁) × f₂-(A₂)
    using ZF_fun_from_tot_val func1_1_L15 by auto
qed
```

For a function of two variables defined on $X \times Y$, if we fix an $x \in X$ we obtain a function on Y. Note that if `domain(f)` is $X \times Y$, `range(domain(f))` extracts Y from $X \times Y$.

```
definition
  Fix1stVar(f,x) ≡ {⟨y,f⟨x,y⟩⟩. y ∈ range(domain(f))}
```

For every $y \in Y$ we can fix the second variable in a binary function $f : X \times Y \to Z$ to get a function on X.

```
definition
  Fix2ndVar(f,y) ≡ {⟨x,f⟨x,y⟩⟩. x ∈ domain(domain(f))}
```

We defined `Fix1stVar` and `Fix2ndVar` so that the domain of the function is not listed in the arguments, but is recovered from the function. The next lemma is a technical fact that makes it easier to use this definition.

```
lemma fix_var_fun_domain: assumes A1: f : X×Y → Z
  shows
  x∈X ⟶ Fix1stVar(f,x) = {⟨y,f⟨x,y⟩⟩. y ∈ Y}
  y∈Y ⟶ Fix2ndVar(f,y) = {⟨x,f⟨x,y⟩⟩. x ∈ X}
proof -
  from A1 have I: domain(f) = X×Y using func1_1_L1 by simp
  { assume x∈X
    with I have range(domain(f)) = Y by auto
    then have Fix1stVar(f,x) = {⟨y,f⟨x,y⟩⟩. y ∈ Y}
      using Fix1stVar_def by simp
  } then show x∈X ⟶ Fix1stVar(f,x) = {⟨y,f⟨x,y⟩⟩. y ∈ Y}
    by simp
  { assume y∈Y
    with I have domain(domain(f)) = X by auto
    then have Fix2ndVar(f,y) = {⟨x,f⟨x,y⟩⟩. x ∈ X}
      using Fix2ndVar_def by simp
  } then show y∈Y ⟶ Fix2ndVar(f,y) = {⟨x,f⟨x,y⟩⟩. x ∈ X}
    by simp
qed
```

If we fix the first variable, we get a function of the second variable.

```
lemma fix_1st_var_fun: assumes A1: f : X×Y → Z and A2: x∈X
  shows Fix1stVar(f,x) : Y → Z
proof -
  from A1 A2 have ∀y∈Y. f⟨x,y⟩ ∈ Z
    using apply_funtype by simp
  then have {⟨y,f⟨x,y⟩⟩. y ∈ Y} :  Y → Z using ZF_fun_from_total by simp
  with A1 A2 show Fix1stVar(f,x) : Y → Z using fix_var_fun_domain by
simp
qed
```

If we fix the second variable, we get a function of the first variable.

```
lemma fix_2nd_var_fun: assumes A1: f : X×Y → Z and A2: y∈Y
  shows Fix2ndVar(f,y) : X → Z
proof -
  from A1 A2 have ∀x∈X. f⟨x,y⟩ ∈ Z
    using apply_funtype by simp
  then have {⟨x,f⟨x,y⟩⟩. x ∈ X} :  X → Z
    using ZF_fun_from_total by simp
  with A1 A2 show Fix2ndVar(f,y) : X → Z
    using fix_var_fun_domain by simp
qed
```

What is the value of `Fix1stVar(f,x)` at $y \in Y$ and the value of `Fix2ndVar(f,y)` at $x \in X$"?

```
lemma fix_var_val:
  assumes A1: f : X×Y → Z and A2: x∈X  y∈Y
  shows
  Fix1stVar(f,x)(y) = f⟨x,y⟩
```

```
  Fix2ndVar(f,y)(x) = f⟨x,y⟩
proof -
  let f₁ = {⟨y,f⟨x,y⟩⟩. y ∈ Y}
  let f₂ = {⟨x,f⟨x,y⟩⟩. x ∈ X}
  from A1 A2 have I:
    Fix1stVar(f,x) = f₁
    Fix2ndVar(f,y) = f₂
    using fix_var_fun_domain by auto
  moreover from A1 A2 have
    Fix1stVar(f,x) : Y → Z
    Fix2ndVar(f,y) : X → Z
    using fix_1st_var_fun fix_2nd_var_fun by auto
  ultimately have f₁ : Y → Z and  f₂ : X → Z
    by auto
  with A2 have f₁(y) = f⟨x,y⟩ and f₂(x) = f⟨x,y⟩
    using ZF_fun_from_tot_val by auto
  with I show
    Fix1stVar(f,x)(y) = f⟨x,y⟩
    Fix2ndVar(f,y)(x) = f⟨x,y⟩
    by auto
qed
```

Fixing the second variable commutes with restrictig the domain.

```
lemma fix_2nd_var_restr_comm:
  assumes A1: f : X×Y → Z and A2: y∈Y and A3: X₁ ⊆ X
  shows Fix2ndVar(restrict(f,X₁×Y),y) = restrict(Fix2ndVar(f,y),X₁)
proof -
  let g = Fix2ndVar(restrict(f,X₁×Y),y)
  let h = restrict(Fix2ndVar(f,y),X₁)
  from A3 have I: X₁×Y ⊆ X×Y by auto
  with A1 have II: restrict(f,X₁×Y) : X₁×Y → Z
    using restrict_type2 by simp
  with A2 have g : X₁ → Z
    using fix_2nd_var_fun by simp
  moreover
  from A1 A2 have III: Fix2ndVar(f,y) : X → Z
    using fix_2nd_var_fun by simp
  with A3 have h : X₁ → Z
    using restrict_type2 by simp
  moreover
  { fix z assume A4: z ∈ X₁
    with A2 I II have g(z) = f⟨z,y⟩
      using restrict fix_var_val by simp
    also from A1 A2 A3 A4 have f⟨z,y⟩ = h(z)
      using restrict fix_var_val by auto
    finally have g(z) = h(z) by simp
  } then have ∀z ∈ X₁. g(z) = h(z) by simp
  ultimately show g = h by (rule func_eq)
qed
```

The next lemma expresses the inverse image of a set by function with fixed first variable in terms of the original function.

```
lemma fix_1st_var_vimage:
  assumes A1: f : X×Y → Z and A2: x∈X
  shows Fix1stVar(f,x)-(A) = {y∈Y. ⟨x,y⟩ ∈ f-(A)}
proof -
  from assms have Fix1stVar(f,x)-(A) = {y∈Y. Fix1stVar(f,x)(y) ∈ A}
    using fix_1st_var_fun func1_1_L15 by blast
  with assms show thesis using fix_var_val func1_1_L15 by auto
qed
```

The next lemma expresses the inverse image of a set by function with fixed second variable in terms of the original function.

```
lemma fix_2nd_var_vimage:
  assumes A1: f : X×Y → Z and A2: y∈Y
  shows Fix2ndVar(f,y)-(A) = {x∈X. ⟨x,y⟩ ∈ f-(A)}
proof -
  from assms have I: Fix2ndVar(f,y)-(A) = {x∈X. Fix2ndVar(f,y)(x) ∈ A}
    using fix_2nd_var_fun func1_1_L15 by blast
  with assms show thesis using fix_var_val func1_1_L15 by auto
qed

end
```

10 Binary operations

```
theory func_ZF imports func1

begin
```

In this theory we consider properties of functions that are binary operations, that is they map $X \times X$ into X.

10.1 Lifting operations to a function space

It happens quite often that we have a binary operation on some set and we need a similar operation that is defined for functions on that set. For example once we know how to add real numbers we also know how to add real-valued functions: for $f, g : X \to \mathbf{R}$ we define $(f + g)(x) = f(x) + g(x)$. Note that formally the $+$ means something different on the left hand side of this equality than on the right hand side. This section aims at formalizing this process. We will call it "lifting to a function space", if you have a suggestion for a better name, please let me know.

Since we are writing in generic set notation, the definition below is a bit complicated. Here it what it says: Given a set X and another set f (that represents a binary function on X) we are defining f lifted to function space over X as the binary function (a set of pairs) on the space $F = X \to \text{range}(f)$ such that the value of this function on pair $\langle a, b\rangle$ of functions on X is another function c on X with values defined by $c(x) = f\langle a(x), b(x)\rangle$.

```
definition
Lift2FcnSpce (infix {lifted to function space over} 65) where
 f {lifted to function space over} X ≡
  {⟨ p,{⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X}⟩.
  p ∈ (X→range(f))×(X→range(f))}
```

The result of the lift belongs to the function space.

```
lemma func_ZF_1_L1:
  assumes A1: f : Y×Y→Y
  and A2: p ∈(X→range(f))×(X→range(f))
  shows
  {⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X} : X→range(f)
  proof -
    have ∀x∈X. f⟨fst(p)(x),snd(p)(x)⟩ ∈ range(f)
    proof
      fix x assume x∈X
      let p = ⟨fst(p)(x),snd(p)(x)⟩
      from A2 ‹x∈X› have
 fst(p)(x) ∈ range(f)  snd(p)(x) ∈ range(f)
 using apply_type by auto
      with A1 have p ∈ Y×Y
 using func1_1_L5B by blast
      with A1 have ⟨p, f(p)⟩ ∈ f
 using apply_Pair by simp
      with A1 show
 f(p) ∈ range(f)
 using rangeI by simp
    qed
    then show thesis using ZF_fun_from_total by simp
qed
```

The values of the lift are defined by the value of the liftee in a natural way.

```
lemma func_ZF_1_L2:
  assumes A1: f : Y×Y→Y
  and A2: p ∈ (X→range(f))×(X→range(f)) and A3: x∈X
  and A4: P = {⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X}
  shows P(x) = f⟨fst(p)(x),snd(p)(x)⟩
proof -
  from A1 A2 have
    {⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X} : X → range(f)
    using func_ZF_1_L1 by simp
  with A4 have P :  X → range(f) by simp
  with  A3 A4 show P(x) = f⟨fst(p)(x),snd(p)(x)⟩
    using ZF_fun_from_tot_val by simp
```

```
qed
```

Function lifted to a function space results in function space operator.

```
theorem func_ZF_1_L3:
  assumes f : Y×Y→Y
  and F = f {lifted to function space over} X
  shows F : (X→range(f))×(X→range(f))→(X→range(f))
  using assms Lift2FcnSpce_def func_ZF_1_L1 ZF_fun_from_total
  by simp
```

The values of the lift are defined by the values of the liftee in the natural way.

```
theorem func_ZF_1_L4:
  assumes A1: f : Y×Y→Y
  and A2: F = f {lifted to function space over} X
  and A3: s:X→range(f) r:X→range(f)
  and A4: x∈X
  shows (F⟨s,r⟩)(x) = f⟨s(x),r(x)⟩
proof -
  let p = ⟨s,r⟩
  let P = {⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X}
  from A1 A3 A4 have
    f : Y×Y→Y  p ∈ (X→range(f))×(X→range(f))
    x∈X  P = {⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X}
    by auto
  then have P(x) = f⟨fst(p)(x),snd(p)(x)⟩
    by (rule func_ZF_1_L2)
  hence P(x) = f⟨s(x),r(x)⟩ by auto
  moreover have P = F⟨s,r⟩
  proof -
    from A1 A2 have F : (X→range(f))×(X→range(f))→(X→range(f))
      using func_ZF_1_L3 by simp
    moreover from A3 have p ∈ (X→range(f))×(X→range(f))
      by auto
    moreover from A2 have
      F = {⟨p,{⟨x,f⟨fst(p)(x),snd(p)(x)⟩⟩. x ∈ X}⟩.
      p ∈ (X→range(f))×(X→range(f))}
      using Lift2FcnSpce_def by simp
    ultimately show thesis using ZF_fun_from_tot_val
      by simp
  qed
  ultimately show (F⟨s,r⟩)(x) = f⟨s(x),r(x)⟩ by auto
qed
```

10.2 Associative and commutative operations

In this section we define associative and commutative operations and prove that they remain such when we lift them to a function space.

Typically we say that a binary operation "$\cdot$" on a set G is "associative" if $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x, y, z \in G$. Our actual definition below does not use the multiplicative notation so that we can apply it equally to the additive notation $+$ or whatever infix symbol we may want to use. Instead, we use the generic set theory notation and write $P\langle x, y\rangle$ to denote the value of the operation P on a pair $\langle x, y\rangle \in G \times G$.

```
definition
  IsAssociative (infix {is associative on} 65) where
  P {is associative on} G ≡ P : G×G→G ∧
  (∀ x ∈ G. ∀ y ∈ G. ∀ z ∈ G.
  ( P(⟨P(⟨x,y⟩),z⟩) = P( ⟨x,P(⟨y,z⟩)⟩ )))
```

A binary function $f : X \times X \to Y$ is commutative if $f\langle x, y\rangle = f\langle y, x\rangle$. Note that in the definition of associativity above we talk about binary "operation" and here we say use the term binary "function". This is not set in stone, but usually the word "operation" is used when the range is a factor of the domain, while the word "function" allows the range to be a completely unrelated set.

```
definition
  IsCommutative (infix {is commutative on} 65) where
  f {is commutative on} G ≡ ∀x∈G. ∀y∈G. f⟨x,y⟩ = f⟨y,x⟩
```

The lift of a commutative function is commutative.

```
lemma func_ZF_2_L1:
  assumes A1: f : G×G→G
  and A2: F = f {lifted to function space over} X
  and A3: s : X→range(f) r : X→range(f)
  and A4: f {is commutative on} G
  shows F⟨s,r⟩ = F⟨r,s⟩
proof -
  from A1 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using func_ZF_1_L3 by simp
  with A3 have
    F⟨s,r⟩ : X→range(f) and F⟨r,s⟩ : X→range(f)
    using apply_type by auto
  moreover have
    ∀x∈X. (F⟨s,r⟩)(x) = (F⟨r,s⟩)(x)
  proof
    fix x assume x∈X
    from A1 have range(f)⊆G
      using func1_1_L5B by simp
    with A3 ‹x∈X› have s(x) ∈ G and r(x) ∈ G
      using apply_type by auto
    with A1 A2 A3 A4 ‹x∈X› show
      (F⟨s,r⟩)(x) = (F⟨r,s⟩)(x)
      using func_ZF_1_L4 IsCommutative_def by simp
```

```
  qed
  ultimately show thesis using fun_extension_iff
    by simp
qed
```

The lift of a commutative function is commutative on the function space.

```
lemma func_ZF_2_L2:
  assumes f : G×G→G
  and f {is commutative on} G
  and F = f {lifted to function space over} X
  shows F {is commutative on} (X→range(f))
  using assms IsCommutative_def func_ZF_2_L1 by simp
```

The lift of an associative function is associative.

```
lemma func_ZF_2_L3:
  assumes A2: F = f {lifted to function space over} X
  and A3: s : X→range(f) r : X→range(f) q : X→range(f)
  and A4: f {is associative on} G
  shows F⟨F⟨s,r⟩,q⟩ = F⟨s,F⟨r,q⟩⟩
proof -
  from A4 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using IsAssociative_def func_ZF_1_L3 by auto
  with A3 have I:
    F⟨s,r⟩ : X→range(f)
    F⟨r,q⟩ : X→range(f)
    F⟨F⟨s,r⟩,q⟩ : X→range(f)
    F⟨s,F⟨r,q⟩⟩: X→range(f)
    using apply_type by auto
  moreover have
    ∀x∈X. (F⟨F⟨s,r⟩,q⟩)(x) = (F⟨s,F⟨r,q⟩⟩)(x)
  proof
    fix x assume x∈X
    from A4 have f:G×G→G
      using IsAssociative_def by simp
    then have range(f)⊆G
      using func1_1_L5B by simp
    with A3 ‹x∈X› have
      s(x) ∈ G r(x) ∈ G q(x) ∈ G
      using apply_type by auto
    with A2 I A3 A4 ‹x∈X› ‹f:G×G→G› show
      (F⟨F⟨s,r⟩,q⟩)(x) = (F⟨s,F⟨r,q⟩⟩)(x)
      using func_ZF_1_L4 IsAssociative_def by simp
  qed
  ultimately show thesis using fun_extension_iff
    by simp
qed
```

The lift of an associative function is associative on the function space.

```
lemma func_ZF_2_L4:
  assumes A1: f {is associative on} G
  and A2: F = f {lifted to function space over} X
  shows F {is associative on} (X→range(f))
proof -
  from A1 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using IsAssociative_def func_ZF_1_L3 by auto
  moreover from A1 A2 have
    ∀s ∈ X→range(f). ∀ r ∈ X→range(f). ∀q ∈ X→range(f).
    F⟨F⟨s,r⟩,q⟩ = F⟨s,F⟨r,q⟩⟩
    using func_ZF_2_L3 by simp
  ultimately show thesis using IsAssociative_def
    by simp
qed
```

10.3 Restricting operations

In this section we consider conditions under which restriction of the operation to a set inherits properties like commutativity and associativity.

The commutativity is inherited when restricting a function to a set.

```
lemma func_ZF_4_L1:
  assumes A1: f:X×X→Y and A2: A⊆X
  and A3: f {is commutative on} X
  shows restrict(f,A×A) {is commutative on} A
proof -
  { fix x y assume x∈A and y∈A
    with A2 have x∈X and y∈X by auto
    with A3 ‹x∈A› ‹y∈A› have
      restrict(f,A×A)⟨x,y⟩ = restrict(f,A×A)⟨y,x⟩
      using IsCommutative_def restrict_if by simp }
  then show thesis using IsCommutative_def by simp
qed
```

Next we define what it means that a set is closed with respect to an operation.

```
definition
  IsOpClosed (infix {is closed under} 65) where
  A {is closed under} f ≡ ∀x∈A. ∀y∈A. f⟨x,y⟩ ∈ A
```

Associative operation restricted to a set that is closed with resp. to this operation is associative.

```
lemma func_ZF_4_L2:assumes A1: f {is associative on} X
  and A2: A⊆X and A3: A {is closed under} f
  and A4: x∈A y∈A z∈A
  and A5: g = restrict(f,A×A)
  shows g⟨g⟨x,y⟩,z⟩ = g⟨x,g⟨y,z⟩⟩
```

```
proof -
  from A4 A2 have I: x∈X y∈X z∈X
    by auto
  from A3 A4 A5 have
    g⟨g⟨x,y⟩,z⟩ = f⟨f⟨x,y⟩,z⟩
    g⟨x,g⟨y,z⟩⟩ = f⟨x,f⟨y,z⟩⟩
    using IsOpClosed_def restrict_if by auto
  moreover from A1 I have
    f⟨f⟨x,y⟩,z⟩ = f⟨x,f⟨y,z⟩⟩
    using IsAssociative_def by simp
  ultimately show thesis by simp
qed
```

An associative operation restricted to a set that is closed with resp. to this operation is associative on the set.

```
lemma func_ZF_4_L3: assumes A1: f {is associative on} X
  and A2: A⊆X and A3: A {is closed under} f
  shows restrict(f,A×A) {is associative on} A
proof -
  let g = restrict(f,A×A)
  from A1 have f:X×X→X
    using IsAssociative_def by simp
  moreover from A2 have A×A ⊆ X×X by auto
  moreover from A3 have ∀p ∈ A×A. g(p) ∈ A
    using IsOpClosed_def restrict_if by auto
  ultimately have g : A×A→A
    using func1_2_L4 by simp
  moreover from  A1 A2 A3 have
    ∀ x ∈ A. ∀ y ∈ A. ∀ z ∈ A.
    g⟨g⟨x,y⟩,z⟩ = g⟨ x,g⟨y,z⟩⟩
    using func_ZF_4_L2 by simp
  ultimately show thesis
    using IsAssociative_def by simp
qed
```

The essential condition to show that if a set A is closed with respect to an operation, then it is closed under this operation restricted to any superset of A.

```
lemma func_ZF_4_L4: assumes A {is closed under} f
  and A⊆B and x∈A  y∈A and g = restrict(f,B×B)
  shows g⟨x,y⟩ ∈ A
  using assms IsOpClosed_def restrict by auto
```

If a set A is closed under an operation, then it is closed under this operation restricted to any superset of A.

```
lemma func_ZF_4_L5:
  assumes A1: A {is closed under} f
  and A2: A⊆B
```

```
  shows A {is closed under} restrict(f,B×B)
proof -
  let g = restrict(f,B×B)
  from A1 A2 have ∀x∈A. ∀y∈A. g⟨x,y⟩ ∈ A
    using func_ZF_4_L4 by simp
  then show thesis using IsOpClosed_def by simp
qed
```

The essential condition to show that intersection of sets that are closed with respect to an operation is closed with respect to the operation.

```
lemma func_ZF_4_L6:
  assumes A {is closed under} f
  and B {is closed under} f
  and x ∈ A∩B y∈ A∩B
  shows f⟨x,y⟩ ∈ A∩B using assms IsOpClosed_def by auto
```

Intersection of sets that are closed with respect to an operation is closed under the operation.

```
lemma func_ZF_4_L7:
  assumes A {is closed under} f
  B {is closed under} f
  shows A∩B {is closed under} f
  using assms IsOpClosed_def by simp
```

10.4 Compositions

For any set X we can consider a binary operation on the set of functions $f : X \to X$ defined by $C(f,g) = f \circ g$. Composition of functions (or relations) is defined in the standard Isabelle distribution as a higher order function and denoted with the letter `O`. In this section we consider the corresponding two-argument ZF-function (binary operation), that is a subset of $((X \to X) \times (X \to X)) \times (X \to X)$.

We define the notion of composition on the set X as the binary operation on the function space $X \to X$ that takes two functions and creates the their composition.

```
definition
  Composition(X) ≡
  {⟨p,fst(p) O snd(p)⟩. p ∈ (X→X)×(X→X)}
```

Composition operation is a function that maps $(X \to X) \times (X \to X)$ into $X \to X$.

```
lemma func_ZF_5_L1: shows Composition(X) : (X→X)×(X→X)→(X→X)
  using comp_fun Composition_def ZF_fun_from_total by simp
```

The value of the composition operation is the composition of arguments.

```
lemma func_ZF_5_L2: assumes f:X→X and g:X→X
```

```
  shows Composition(X)⟨f,g⟩ = f O g
proof -
  from assms have
    Composition(X) : (X→X)×(X→X)→(X→X)
    ⟨f,g⟩ ∈ (X→X)×(X→X)
    Composition(X) = {⟨p,fst(p) O snd(p)⟩. p ∈ (X→X)×(X→X)}
    using  func_ZF_5_L1 Composition_def by auto
  then show Composition(X)⟨f,g⟩ = f O g
    using  ZF_fun_from_tot_val by auto
qed
```

What is the value of a composition on an argument?

```
lemma func_ZF_5_L3: assumes f:X→X and g:X→X and x∈X
  shows (Composition(X)⟨f,g⟩)(x) = f(g(x))
  using assms func_ZF_5_L2 comp_fun_apply by simp
```

The essential condition to show that composition is associative.

```
lemma func_ZF_5_L4: assumes A1: f:X→X g:X→X h:X→X
  and A2: C = Composition(X)
  shows C⟨C⟨f,g⟩,h⟩ = C⟨ f,C⟨g,h⟩⟩
proof -
  from A2 have C : ((X→X)×(X→X))→(X→X)
    using func_ZF_5_L1 by simp
  with A1 have I:
    C⟨f,g⟩ : X→X
    C⟨g,h⟩ : X→X
    C⟨C⟨f,g⟩,h⟩ : X→X
    C⟨ f,C⟨g,h⟩⟩ : X→X
    using apply_funtype by auto
  moreover have
    ∀ x ∈ X. C⟨C⟨f,g⟩,h⟩(x) = C⟨f,C⟨g,h⟩⟩(x)
  proof
    fix x assume x∈X
    with A1 A2 I have
      C⟨C⟨f,g⟩,h⟩  (x) = f(g(h(x)))
      C⟨ f,C⟨g,h⟩⟩(x) = f(g(h(x)))
      using func_ZF_5_L3 apply_funtype by auto
    then show C⟨C⟨f,g⟩,h⟩(x) = C⟨ f,C⟨g,h⟩⟩(x)
      by simp
    qed
  ultimately show thesis using fun_extension_iff by simp
qed
```

Composition is an associative operation on $X \to X$ (the space of functions that map X into itself).

```
lemma func_ZF_5_L5: shows Composition(X) {is associative on} (X→X)
proof -
  let C = Composition(X)
  have ∀f∈X→X. ∀g∈X→X. ∀h∈X→X.
```

```
    C⟨C⟨f,g⟩,h⟩ = C⟨f,C⟨g,h⟩⟩
    using func_ZF_5_L4 by simp
  then show thesis using func_ZF_5_L1 IsAssociative_def
    by simp
qed
```

10.5 Identity function

In this section we show some additional facts about the identity function defined in the standard Isabelle's `Perm` theory. Note there is also `image_id_same` lemma in `func1` theory.

A function that maps every point to itself is the identity on its domain.

```
lemma indentity_fun: assumes A1: f:X→Y and A2:∀x∈X. f(x)=x
  shows f = id(X)
proof -
  from assms have f:X→Y and id(X):X→X and ∀x∈X. f(x) = id(X)(x)
    using id_type id_conv by auto
  then show thesis by (rule func_eq)
qed
```

Composing a function with identity does not change the function.

```
lemma func_ZF_6_L1A: assumes A1: f : X→X
  shows Composition(X)⟨f,id(X)⟩ = f
  Composition(X)⟨id(X),f⟩ = f
proof -
  have Composition(X) : (X→X)×(X→X)→(X→X)
    using func_ZF_5_L1 by simp
  with A1 have Composition(X)⟨id(X),f⟩ : X→X
    Composition(X)⟨f,id(X)⟩ : X→X
    using id_type apply_funtype by auto
  moreover note A1
  moreover from A1 have
    ∀x∈X. (Composition(X)⟨id(X),f⟩)(x) = f(x)
    ∀x∈X. (Composition(X)⟨f,id(X)⟩)(x) = f(x)
    using id_type func_ZF_5_L3 apply_funtype id_conv
    by auto
  ultimately show Composition(X)⟨id(X),f⟩ = f
    Composition(X)⟨f,id(X)⟩ = f
    using fun_extension_iff by auto
qed
```

An intuitively clear, but surprisingly nontrivial fact: identity is the only function from a singleton to itself.

```
lemma singleton_fun_id: shows ({x} → {x}) = {id({x})}
proof
  show {id({x})} ⊆ ({x} → {x})
    using id_def by simp
```

```
  { let g = id({x})
    fix f assume f : {x} → {x}
    then have f : {x} → {x} and g : {x} → {x}
      using id_def by auto
    moreover from ‹f : {x} → {x}› have ∀x ∈ {x}. f(x) = g(x)
      using apply_funtype id_def by auto
    ultimately have f = g by (rule func_eq)
  } then show  ({x} → {x}) ⊆ {id({x})} by auto
qed
```

Another trivial fact: identity is the only bijection of a singleton with itself.

```
lemma single_bij_id: shows bij({x},{x}) = {id({x})}
proof
  show {id({x})} ⊆ bij({x},{x}) using id_bij
    by simp
  { fix f assume f ∈ bij({x},{x})
    then have f : {x} → {x} using bij_is_fun
      by simp
    then have f ∈ {id({x})} using singleton_fun_id
      by simp
  } then show bij({x},{x}) ⊆ {id({x})} by auto
qed
```

A kind of induction for the identity: if a function f is the identity on a set with a fixpoint of f removed, then it is the indentity on the whole set.

```
lemma id_fixpoint_rem: assumes A1: f:X→X and
  A2: p∈X and A3: f(p) = p and
  A4: restrict(f, X-{p}) = id(X-{p})
  shows f = id(X)
proof -
  from A1 have f: X→X and id(X) : X→X
    using id_def by auto
  moreover
  { fix x assume x∈X
    { assume x ∈ X-{p}
      then have f(x) = restrict(f, X-{p})(x)
 using restrict by simp
      with A4 ‹x ∈ X-{p}› have f(x) = x
 using id_def by simp }
    with A2 A3 ‹x∈X› have f(x) = x by auto
  } then have ∀x∈X. f(x) = id(X)(x)
    using id_def by simp
  ultimately show f = id(X) by (rule func_eq)
qed
```

10.6 Lifting to subsets

Suppose we have a binary operation $f : X \times X \to X$ written additively as $f\langle x, y\rangle = x + y$. Such operation naturally defines another binary operation on the subsets of X that satisfies $A+B=\{x+y: x\in A, y\in B\}$. This new operation which we will call "f lifted to subsets" inherits many properties of f, such as associativity, commutativity and existence of the neutral element. This notion is useful for considering interval arithmetics.

The next definition describes the notion of a binary operation lifted to subsets. It is written in a way that might be a bit unexpected, but really it is the same as the intuitive definition, but shorter. In the definition we take a pair $p\in Pow(X)\times Pow(X)$, say $p=\langle A,B\rangle$, where $A,B\subseteq X$. Then we assign this pair of sets the set $\{f\langle x,y\rangle : x\in A, y\in B\} = \{f(x') : x'\in A\times B\}$ The set on the right hand side is the same as the image of $A\times B$ under f. In the definition we don't use A and B symbols, but write `fst(p)` and `snd(p)`, resp. Recall that in Isabelle/ZF `fst(p)` and `snd(p)` denote the first and second components of an ordered pair p. See the lemma `lift_subsets_explained` for a more intuitive notation.

```
definition
  Lift2Subsets (infix {lifted to subsets of} 65) where
  f {lifted to subsets of} X ≡
  {⟨p, f(fst(p)×snd(p))⟩. p ∈ Pow(X)×Pow(X)}
```

The lift to subsets defines a binary operation on the subsets.

```
lemma lift_subsets_binop: assumes A1: f : X × X → Y
  shows (f {lifted to subsets of} X) : Pow(X) × Pow(X) → Pow(Y)
proof -
  let F = {⟨p, f(fst(p)×snd(p))⟩. p ∈ Pow(X)×Pow(X)}
  from A1 have ∀p ∈ Pow(X) × Pow(X). f(fst(p)×snd(p)) ∈ Pow(Y)
    using func1_1_L6 by simp
  then have F : Pow(X) × Pow(X) → Pow(Y)
    by (rule ZF_fun_from_total)
  then show thesis unfolding Lift2Subsets_def by simp
qed
```

The definition of the lift to subsets rewritten in a more intuitive notation. We would like to write the last assertion as `F⟨A,B⟩ = {f⟨x,y⟩. x ∈ A, y ∈ B}`, but Isabelle/ZF does not allow such syntax.

```
lemma lift_subsets_explained: assumes A1: f : X×X → Y
  and A2: A ⊆ X  B ⊆ X and A3: F = f {lifted to subsets of} X
  shows
  F⟨A,B⟩ ⊆ Y and
  F⟨A,B⟩ = f(A×B)
  F⟨A,B⟩ = {f(p). p ∈ A×B}
  F⟨A,B⟩ = {f⟨x,y⟩ . ⟨x,y⟩ ∈ A×B}
proof -
  let p = ⟨A,B⟩
  from assms have
    I: F : Pow(X) × Pow(X) → Pow(Y) and  p ∈ Pow(X) × Pow(X)
    using lift_subsets_binop by auto
```

```
  moreover from A3 have F = {⟨p, f(fst(p)×snd(p))⟩. p ∈ Pow(X)×Pow(X)}
    unfolding  Lift2Subsets_def by simp
  ultimately show F⟨A,B⟩ =  f(A×B)
    using  ZF_fun_from_tot_val by auto
  also
  from A1 A2 have A×B ⊆ X×X by auto
  with A1 have f(A×B) = {f(p). p ∈ A×B}
    by (rule func_imagedef)
  finally show  F⟨A,B⟩ = {f(p) . p ∈ A×B} by simp
  also
  have ∀x∈A. ∀y ∈ B. f⟨x,y⟩ = f⟨x,y⟩ by simp
  then have {f(p). p ∈ A×B} = {f⟨x,y⟩.  ⟨x,y⟩ ∈ A×B}
    by (rule ZF1_1_L4A)
  finally show F⟨A,B⟩ = {f⟨x,y⟩ . ⟨x,y⟩ ∈ A×B}
    by simp
  from A2 I show F⟨A,B⟩ ⊆ Y using apply_funtype by blast
qed
```

A sufficient condition for a point to belong to a result of lifting to subsets.

```
lemma lift_subset_suff:  assumes A1: f : X × X → Y and
  A2: A ⊆ X  B ⊆ X and A3: x∈A y∈B and
  A4: F = f {lifted to subsets of} X
  shows f⟨x,y⟩ ∈ F⟨A,B⟩
proof -
  from A3 have f⟨x,y⟩ ∈ {f(p) . p ∈ A×B} by auto
  moreover from A1 A2 A4 have {f(p). p ∈ A×B} = F⟨A,B⟩
    using lift_subsets_explained by simp
  ultimately show f⟨x,y⟩ ∈ F⟨A,B⟩ by simp
qed
```

A kind of converse of `lift_subset_apply`, providing a necessary condition for a point to be in the result of lifting to subsets.

```
lemma lift_subset_nec: assumes A1: f : X × X → Y and
  A2: A ⊆ X  B ⊆ X and
  A3: F = f {lifted to subsets of} X and
  A4: z ∈ F⟨A,B⟩
  shows ∃x y. x∈A ∧ y∈B ∧ z = f⟨x,y⟩
proof -
  from A1 A2 A3 have F⟨A,B⟩ = {f(p). p ∈ A×B}
    using lift_subsets_explained by simp
  with A4 show thesis by auto
qed
```

Lifting to subsets inherits commutativity.

```
lemma lift_subset_comm: assumes A1: f : X × X → Y and
  A2: f {is commutative on} X and
  A3: F = f {lifted to subsets of} X
  shows F {is commutative on} Pow(X)
proof -
```

```
  have ∀A ∈ Pow(X). ∀B ∈ Pow(X). F⟨A,B⟩ = F⟨B,A⟩
  proof -
    { fix A assume A ∈ Pow(X)
      fix B assume B ∈ Pow(X)
      have  F⟨A,B⟩ = F⟨B,A⟩
      proof -
 have ∀z ∈  F⟨A,B⟩. z ∈  F⟨B,A⟩
 proof
   fix z assume I: z ∈ F⟨A,B⟩
   with A1 A3 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have
     ∃x y. x∈A ∧ y∈B ∧ z = f⟨x,y⟩
     using lift_subset_nec by simp
   then obtain x y where x∈A and y∈B and z = f⟨x,y⟩
     by auto
   with A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have z = f⟨y,x⟩
     using IsCommutative_def by auto
   with A1 A3 I ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹x∈A› ‹y∈B›
   show z ∈ F⟨B,A⟩ using lift_subset_suff by simp
 qed
 moreover have ∀z ∈  F⟨B,A⟩. z ∈  F⟨A,B⟩
 proof
   fix z assume I: z ∈ F⟨B,A⟩
   with A1 A3 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have
     ∃x y. x∈B ∧ y∈A ∧ z = f⟨x,y⟩
     using lift_subset_nec by simp
   then obtain x y where x∈B and y∈A and z = f⟨x,y⟩
     by auto
   with A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› have z = f⟨y,x⟩
     using IsCommutative_def by auto
   with A1 A3 I ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹x∈B› ‹y∈A›
   show z ∈ F⟨A,B⟩ using lift_subset_suff by simp
 qed
 ultimately show F⟨A,B⟩ = F⟨B,A⟩ by auto
      qed
    } thus thesis by auto
  qed
  then show F {is commutative on} Pow(X)
    unfolding IsCommutative_def by auto
qed
```

Lifting to subsets inherits associativity. To show that $F\langle\langle A,B\rangle C\rangle = F\langle A,F\langle B,C\rangle\rangle$ we prove two inclusions and the proof of the second inclusion is very similar to the proof of the first one.

```
lemma lift_subset_assoc:  assumes
  A1: f {is associative on} X and A2: F = f {lifted to subsets of} X
  shows F {is associative on} Pow(X)
proof -
  from A1 have f : X×X → X unfolding IsAssociative_def by simp
  with A2 have F : Pow(X)×Pow(X) → Pow(X)
```

```
    using lift_subsets_binop by simp
  moreover have ∀A ∈ Pow(X).∀B ∈ Pow(X). ∀C ∈ Pow(X).
    F⟨F⟨A,B⟩,C⟩ = F⟨A,F⟨B,C⟩⟩
  proof -
    { fix A B C
      assume A ∈ Pow(X)  B ∈ Pow(X)  C ∈ Pow(X)
      have F⟨F⟨A,B⟩,C⟩ ⊆ F⟨A,F⟨B,C⟩⟩
      proof
fix z assume I: z ∈ F⟨F⟨A,B⟩,C⟩
from ‹f:X×X → X› A2 ‹A ∈ Pow(X)›  ‹B ∈ Pow(X)›
have F⟨A,B⟩ ∈ Pow(X)
  using lift_subsets_binop apply_funtype by blast
with ‹f:X×X → X› A2 ‹C ∈ Pow(X)› I have
  ∃x y. x ∈ F⟨A,B⟩ ∧ y ∈ C ∧ z = f⟨x,y⟩
  using lift_subset_nec by simp
then obtain x y where
  II: x ∈ F⟨A,B⟩ and y ∈ C and III: z = f⟨x,y⟩
  by auto
from ‹f:X×X → X› A2 ‹A ∈ Pow(X)›  ‹B ∈ Pow(X)› II have
  ∃ s t. s ∈ A ∧ t ∈ B ∧ x = f⟨s,t⟩
  using lift_subset_nec by auto
then obtain s t where s∈A and t∈B and x = f⟨s,t⟩
  by auto
with A1 ‹A ∈ Pow(X)›  ‹B ∈ Pow(X)› ‹C ∈ Pow(X)› III
  ‹s∈A› ‹t∈B› ‹y∈C› have IV: z = f⟨s, f⟨t,y⟩⟩
  using IsAssociative_def by blast
from ‹f:X×X → X› A2 ‹B ∈ Pow(X)›  ‹C ∈ Pow(X)›  ‹t∈B›  ‹y∈C›
have f⟨t,y⟩ ∈ F⟨B,C⟩ using lift_subset_suff by simp
moreover from ‹f:X×X → X› A2 ‹B ∈ Pow(X)›  ‹C ∈ Pow(X)›
have F⟨B,C⟩ ⊆ X using lift_subsets_binop apply_funtype
  by blast
moreover note ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹s∈A› IV
ultimately show z ∈ F⟨A,F⟨B,C⟩⟩
  using lift_subset_suff by simp
      qed
      moreover have F⟨A,F⟨B,C⟩⟩ ⊆ F⟨F⟨A,B⟩,C⟩
      proof
fix z assume I: z ∈ F⟨A,F⟨B,C⟩⟩
from ‹f:X×X → X› A2 ‹B ∈ Pow(X)› ‹C ∈ Pow(X)›
have F⟨B,C⟩ ∈ Pow(X)
  using lift_subsets_binop apply_funtype by blast
with ‹f:X×X → X› A2 ‹A ∈ Pow(X)› I have
  ∃x y. x ∈ A ∧ y ∈ F⟨B,C⟩ ∧ z = f⟨x,y⟩
  using lift_subset_nec by simp
then obtain x y where
  x ∈ A and II: y ∈ F⟨B,C⟩ and III: z = f⟨x,y⟩
  by auto
from ‹f:X×X → X› A2 ‹B ∈ Pow(X)›  ‹C ∈ Pow(X)› II have
  ∃ s t. s ∈ B ∧ t ∈ C ∧ y = f⟨s,t⟩
```

```
    using lift_subset_nec by auto
  then obtain s t where s∈B and t∈C and y = f⟨s,t⟩
    by auto
  with III have z = f⟨x,f⟨s,t⟩⟩ by simp
  moreover from A1 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹C ∈ Pow(X)›
    ‹x∈A› ‹s∈B› ‹t∈C› have f⟨f⟨x,s⟩,t⟩ = f⟨x,f⟨s,t⟩⟩
    using IsAssociative_def by blast
  ultimately have IV: z = f⟨f⟨x,s⟩,t⟩ by simp
  from ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)› ‹x∈A› ‹s∈B›
  have f⟨x,s⟩ ∈ F⟨A,B⟩ using lift_subset_suff by simp
  moreover from ‹f:X×X → X› A2 ‹A ∈ Pow(X)› ‹B ∈ Pow(X)›
  have F⟨A,B⟩ ⊆ X using lift_subsets_binop apply_funtype
    by blast
  moreover note ‹f:X×X → X› A2 ‹C ∈ Pow(X)› ‹t∈C› IV
  ultimately show z ∈ F⟨F⟨A,B⟩,C⟩
    using lift_subset_suff by simp
      qed
      ultimately have F⟨F⟨A,B⟩,C⟩ = F⟨A,F⟨B,C⟩⟩ by auto
    } thus thesis by auto
  qed
  ultimately show thesis unfolding IsAssociative_def
    by auto
qed
```

10.7 Distributive operations

In this section we deal with pairs of operations such that one is distributive with respect to the other, that is $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(b+c) \cdot a = b \cdot a + c \cdot a$. We show that this property is preserved under restriction to a set closed with respect to both operations. In EquivClass1 theory we show that this property is preserved by projections to the quotient space if both operations are congruent with respect to the equivalence relation.

We define distributivity as a statement about three sets. The first set is the set on which the operations act. The second set is the additive operation (a ZF function) and the third is the multiplicative operation.

```
definition
  IsDistributive(X,A,M) ≡ (∀a∈X.∀b∈X.∀c∈X.
  M⟨a,A⟨b,c⟩⟩ = A⟨M⟨a,b⟩,M⟨a,c⟩⟩ ∧
  M⟨A⟨b,c⟩,a⟩ = A⟨M⟨b,a⟩,M⟨c,a⟩ ⟩)
```

The essential condition to show that distributivity is preserved by restrictions to sets that are closed with respect to both operations.

```
lemma func_ZF_7_L1:
  assumes A1: IsDistributive(X,A,M)
  and A2: Y⊆X
  and A3: Y {is closed under} A  Y {is closed under} M
  and A4: Ar = restrict(A,Y×Y) Mr = restrict(M,Y×Y)
```

```
  and A5: a∈Y  b∈Y  c∈Y
  shows Mr⟨ a,Ar⟨b,c⟩ ⟩  = Ar⟨ Mr⟨a,b⟩,Mr⟨a,c⟩ ⟩  ∧
  Mr⟨ Ar⟨b,c⟩,a ⟩ = Ar⟨ Mr⟨b,a⟩, Mr⟨c,a⟩ ⟩
proof -
  from A3 A5 have A⟨b,c⟩ ∈ Y  M⟨a,b⟩ ∈ Y  M⟨a,c⟩ ∈ Y
    M⟨b,a⟩ ∈ Y  M⟨c,a⟩ ∈ Y using IsOpClosed_def by auto
  with A5 A4 have
    Ar⟨b,c⟩ ∈ Y  Mr⟨a,b⟩ ∈ Y  Mr⟨a,c⟩ ∈ Y
    Mr⟨b,a⟩ ∈ Y  Mr⟨c,a⟩ ∈ Y
    using restrict by auto
  with A1 A2 A4 A5 show thesis
    using restrict IsDistributive_def by auto
qed
```

Distributivity is preserved by restrictions to sets that are closed with respect to both operations.

```
lemma func_ZF_7_L2:
  assumes IsDistributive(X,A,M)
  and Y⊆X
  and Y {is closed under} A
  Y {is closed under} M
  and Ar = restrict(A,Y×Y) Mr = restrict(M,Y×Y)
  shows IsDistributive(Y,Ar,Mr)
proof -
  from assms have ∀a∈Y.∀b∈Y.∀c∈Y.
    Mr⟨ a,Ar⟨b,c⟩ ⟩ = Ar⟨ Mr⟨a,b⟩,Mr⟨a,c⟩ ⟩ ∧
    Mr⟨ Ar⟨b,c⟩,a ⟩ = Ar⟨ Mr⟨b,a⟩,Mr⟨c,a⟩⟩
    using func_ZF_7_L1 by simp
  then show thesis using IsDistributive_def by simp
qed


end
```

11 More on functions

```
theory func_ZF_1 imports ZF.Order Order_ZF_1a func_ZF

begin
```

In this theory we consider some properties of functions related to order relations

11.1 Functions and order

This section deals with functions between ordered sets.

If every value of a function on a set is bounded below by a constant, then the image of the set is bounded below.

```
lemma func_ZF_8_L1:
  assumes f:X→Y and A⊆X and ∀x∈A. ⟨L,f(x)⟩ ∈ r
  shows IsBoundedBelow(f(A),r)
proof -
  from assms have ∀y ∈ f(A). ⟨L,y⟩ ∈ r
    using func_imagedef by simp
  then show IsBoundedBelow(f(A),r)
    by (rule Order_ZF_3_L9)
qed
```

If every value of a function on a set is bounded above by a constant, then the image of the set is bounded above.

```
lemma func_ZF_8_L2:
  assumes f:X→Y and A⊆X and ∀x∈A. ⟨f(x),U⟩ ∈ r
  shows IsBoundedAbove(f(A),r)
proof -
  from assms have ∀y ∈ f(A). ⟨y,U⟩ ∈ r
    using func_imagedef by simp
  then show IsBoundedAbove(f(A),r)
    by (rule Order_ZF_3_L10)
qed
```

Identity is an order isomorphism.

```
lemma id_ord_iso: shows id(X) ∈ ord_iso(X,r,X,r)
  using id_bij id_def ord_iso_def by simp
```

Identity is the only order automorphism of a singleton.

```
lemma id_ord_auto_singleton:
  shows ord_iso({x},r,{x},r) = {id({x})}
  using id_ord_iso ord_iso_def single_bij_id
  by auto
```

The image of a maximum by an order isomorphism is a maximum. Note that from the fact the r is antisymmetric and f is an order isomorphism between (A,r) and (B,R) we can not conclude that R is antisymmetric (we can only show that $R \cap (B \times B)$ is).

```
lemma max_image_ord_iso:
  assumes A1: antisym(r) and A2: antisym(R) and
  A3: f ∈ ord_iso(A,r,B,R) and
  A4: HasAmaximum(r,A)
  shows HasAmaximum(R,B) and Maximum(R,B) = f(Maximum(r,A))
proof -
  let M = Maximum(r,A)
  from A1 A4 have M ∈ A using Order_ZF_4_L3 by simp
  from A3 have f:A→B using ord_iso_def bij_is_fun
    by simp
  with ‹M ∈ A› have I: f(M) ∈ B
    using apply_funtype by simp
```

```
  { fix y assume y ∈ B
    let x = converse(f)(y)
    from A3 have converse(f) ∈ ord_iso(B,R,A,r)
      using ord_iso_sym by simp
    then have converse(f): B → A
      using ord_iso_def bij_is_fun by simp
    with ‹y ∈ B› have x ∈ A
      by simp
    with A1 A3 A4 ‹x ∈ A› ‹M ∈ A› have ⟨f(x), f(M)⟩ ∈ R
      using Order_ZF_4_L3 ord_iso_apply by simp
    with A3 ‹y ∈ B› have ⟨y, f(M)⟩ ∈ R
      using right_inverse_bij ord_iso_def by auto
  } then have II: ∀y ∈ B. ⟨y, f(M)⟩ ∈ R by simp
  with A2 I show Maximum(R,B) = f(M)
    by (rule Order_ZF_4_L14)
  from I II show HasAmaximum(R,B)
    using HasAmaximum_def by auto
qed
```

Maximum is a fixpoint of order automorphism.

```
lemma max_auto_fixpoint:
  assumes antisym(r) and f ∈ ord_iso(A,r,A,r)
  and HasAmaximum(r,A)
  shows Maximum(r,A) = f(Maximum(r,A))
  using assms max_image_ord_iso by blast
```

If two sets are order isomorphic and we remove x and $f(x)$, respectively, from the sets, then they are still order isomorphic.

```
lemma ord_iso_rem_point:
  assumes A1: f ∈ ord_iso(A,r,B,R) and A2: a ∈ A
  shows restrict(f,A-{a}) ∈ ord_iso(A-{a},r,B-{f(a)},R)
proof -
  let f0 = restrict(f,A-{a})
  have A-{a} ⊆ A by auto
  with A1 have f0 ∈ ord_iso(A-{a},r,f(A-{a}),R)
    using ord_iso_restrict_image by simp
  moreover
  from A1 have f ∈ inj(A,B)
    using ord_iso_def bij_def by simp
  with A2  have f(A-{a}) = f(A) - f{a}
    using inj_image_dif by simp
  moreover from A1 have f(A) = B
    using ord_iso_def bij_def surj_range_image_domain
    by auto
  moreover
  from A1 have f: A→B
    using ord_iso_def bij_is_fun by simp
  with A2 have f{a} = {f(a)}
    using singleton_image by simp
```

```
  ultimately show thesis by simp
qed
```

If two sets are order isomorphic and we remove maxima from the sets, then they are still order isomorphic.

```
corollary ord_iso_rem_max:
  assumes A1: antisym(r) and f ∈ ord_iso(A,r,B,R) and
  A4: HasAmaximum(r,A) and  A5: M = Maximum(r,A)
  shows restrict(f,A-{M}) ∈ ord_iso(A-{M}, r, B-{f(M)},R)
  using assms Order_ZF_4_L3 ord_iso_rem_point by simp
```

Lemma about extending order isomorphisms by adding one point to the domain.

```
lemma ord_iso_extend:  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: M_A ∉ A M_B ∉ B and
  A3: ∀a∈A. ⟨a, M_A⟩ ∈ r  ∀b∈B. ⟨b, M_B⟩ ∈ R and
  A4: antisym(r)  antisym(R) and
  A5: ⟨M_A,M_A⟩ ∈ r ⟷ ⟨M_B,M_B⟩ ∈ R
  shows f ∪ {⟨ M_A,M_B⟩} ∈ ord_iso(A∪{M_A} ,r,B∪{M_B} ,R)
proof -
  let g = f ∪ {⟨ M_A,M_B⟩}
  from A1 A2 have
    g : A∪{M_A} → B∪{M_B} and
    I: ∀x∈A. g(x) = f(x) and II: g(M_A) = M_B
    using ord_iso_def bij_def inj_def func1_1_L11D
    by auto
  from A1 A2 have g ∈ bij(A∪{M_A},B∪{M_B})
    using ord_iso_def bij_extend_point by simp
  moreover have ∀x ∈ A∪{M_A}. ∀ y ∈ A∪{M_A}.
    ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
  proof -
    { fix x y
      assume x ∈ A∪{M_A} and y ∈ A∪{M_A}
      then have x∈A ∧ y ∈ A ∨ x∈A ∧ y = M_A ∨
 x = M_A ∧ y ∈ A ∨ x = M_A ∧ y = M_A
 by auto
      moreover
      { assume x∈A ∧ y ∈ A
 with A1 I have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
   using ord_iso_def by simp }
      moreover
      { assume x∈A ∧ y = M_A
 with A1 A3 I II have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
   using ord_iso_def bij_def inj_def apply_funtype
   by auto }
      moreover
      { assume x = M_A ∧ y ∈ A
 with A2 A3 A4 have ⟨x,y⟩ ∉ r
   using antisym_def by auto
```

```
 moreover
 { assume A6: ⟨g(x), g(y)⟩ ∈ R
   from A1 I II ‹x = M_A ∧ y ∈ A› have
     III: g(y) ∈ B  g(x) = M_B
     using ord_iso_def bij_def inj_def apply_funtype
     by auto
   with A3 have ⟨g(y), g(x)⟩ ∈ R by simp
   with A4 A6 have g(y) = g(x) using antisym_def
     by auto
   with A2 III have False by simp
 } hence ⟨g(x), g(y)⟩ ∉ R by auto
 ultimately have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
 by simp }
      moreover
      { assume x = M_A ∧ y = M_A
 with A5 II have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
   by simp }
      ultimately have ⟨x,y⟩ ∈ r ⟷ ⟨g(x), g(y)⟩ ∈ R
 by auto
    } thus thesis by auto
  qed
  ultimately show thesis using ord_iso_def
    by simp
qed
```

A kind of converse to `ord_iso_rem_max`: if two linearly ordered sets sets are order isomorphic after removing the maxima, then they are order isomorphic.

```
lemma rem_max_ord_iso:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: HasAmaximum(r,X)  HasAmaximum(R,Y)
  ord_iso(X - {Maximum(r,X)},r,Y - {Maximum(R,Y)},R) ≠ 0
  shows ord_iso(X,r,Y,R) ≠ 0
proof -
  let M_A = Maximum(r,X)
  let A = X - {M_A}
  let M_B = Maximum(R,Y)
  let B = Y - {M_B}
  from A2 obtain f where f ∈ ord_iso(A,r,B,R)
    by auto
  moreover have M_A ∉ A and M_B ∉ B
    by auto
  moreover from A1 A2 have
    ∀a∈A. ⟨a,M_A⟩ ∈ r and ∀b∈B. ⟨b,M_B⟩ ∈ R
    using IsLinOrder_def Order_ZF_4_L3 by auto
  moreover from A1 have antisym(r) and antisym(R)
    using IsLinOrder_def by auto
  moreover from A1 A2 have ⟨M_A,M_A⟩ ∈ r ⟷ ⟨M_B,M_B⟩ ∈ R
    using IsLinOrder_def Order_ZF_4_L3 IsLinOrder_def
```

```
    total_is_refl refl_def by auto
  ultimately have
    f ∪ {⟨ M_A,M_B⟩} ∈ ord_iso(A∪{M_A} ,r,B∪{M_B} ,R)
    by (rule ord_iso_extend)
  moreover from A1 A2 have
    A∪{M_A} = X and B∪{M_B} = Y
  using IsLinOrder_def Order_ZF_4_L3 by auto
  ultimately show ord_iso(X,r,Y,R) ≠ 0
    using ord_iso_extend by auto
qed
```

11.2 Functions in cartesian products

In this section we consider maps arising naturally in cartesian products.

There is a natural bijection etween $X = Y \times \{y\}$ (a "slice") and Y. We will call this the `SliceProjection(Y×{y})`. This is really the ZF equivalent of the meta-function `fst(x)`.

```
definition
  SliceProjection(X) ≡ {⟨p,fst(p)⟩. p ∈ X }
```

A slice projection is a bijection between $X \times \{y\}$ and X.

```
lemma slice_proj_bij: shows
  SliceProjection(X×{y}): X×{y} → X
  domain(SliceProjection(X×{y})) = X×{y}
  ∀p∈X×{y}. SliceProjection(X×{y})(p) = fst(p)
  SliceProjection(X×{y}) ∈ bij(X×{y},X)
proof -
  let P = SliceProjection(X×{y})
  have  ∀p ∈ X×{y}. fst(p) ∈ X by simp
  moreover from this have
    {⟨p,fst(p)⟩. p ∈ X×{y} } : X×{y} → X
    by (rule ZF_fun_from_total)
  ultimately show
    I: P: X×{y} → X and II: ∀p∈X×{y}. P(p) = fst(p)
    using ZF_fun_from_tot_val SliceProjection_def by auto
  hence
    ∀a ∈ X×{y}. ∀ b ∈ X×{y}. P(a) = P(b) ⟶ a=b
    by auto
  with I have P ∈ inj(X×{y},X) using inj_def
    by simp
  moreover from II have ∀x∈X. ∃p∈X×{y}. P(p) = x
    by simp
  with I have P ∈ surj(X×{y},X) using surj_def
    by simp
  ultimately show P ∈ bij(X×{y},X)
    using bij_def by simp
  from I show domain(SliceProjection(X×{y})) = X×{y}
    using func1_1_L1 by simp
```

qed

Given 2 functions $f : A \to B$ and $g : C \to D$, we can consider a function $h : A \times C \to B \times D$ such that $h(x, y) = \langle f(x), g(y)\rangle$

```
definition
  ProdFunction where
  ProdFunction(f,g) ≡ {⟨z,⟨f(fst(z)),g(snd(z))⟩⟩. z∈domain(f)×domain(g)}
```

For given functions $f : A \to B$ and $g : C \to D$ the function `ProdFunction(f,g)` maps $A \times C$ to $B \times D$.

```
lemma prodFunction:
  assumes f:A→B g:C→D
  shows ProdFunction(f,g):(A×C)→(B×D)
proof-
  from assms have ∀z ∈ domain(f)×domain(g). ⟨f(fst(z)),g(snd(z))⟩ ∈
B×D
    using func1_1_L1 apply_type by auto
  with assms show thesis unfolding ProdFunction_def using func1_1_L1
ZF_fun_from_total
    by simp
qed
```

For given functions $f : A \to B$ and $g : C \to D$ and points $x \in A$, $y \in C$ the value of the function `ProdFunction(f,g)` on $\langle x, y\rangle$ is $\langle f(x), g(y)\rangle$.

```
lemma prodFunctionApp:
  assumes f:A→B g:C→D x∈A y∈C
  shows ProdFunction(f,g)⟨x,y⟩ = ⟨f(x),g(y)⟩
proof -
  let z = ⟨x,y⟩
  from assms have z ∈ A×C and ProdFunction(f,g):(A×C)→(B×D)
    using prodFunction by auto
  moreover from assms(1,2) have ProdFunction(f,g) = {⟨z,⟨f(fst(z)),g(snd(z))⟩⟩.
z∈A×C}
    unfolding ProdFunction_def using func1_1_L1 by blast
  ultimately show thesis using ZF_fun_from_tot_val by auto
qed
```

Somewhat technical lemma about inverse image of a set by a `ProdFunction(f,f)`.

```
lemma prodFunVimage: assumes x∈X f:X→Y
  shows ⟨x,t⟩ ∈ ProdFunction(f,f)-(V) ⟷ t∈X ∧ ⟨fx,ft⟩ ∈ V
proof -
  from assms(2) have T:ProdFunction(f,f)-(V) = {z ∈ X×X. ProdFunction(f,f)(z)
∈ V}
    using prodFunction func1_1_L15 by blast
  with assms show thesis using prodFunctionApp by auto
qed
```

11.3 Induced relations and order isomorphisms

When we have two sets X, Y, function $f : X \to Y$ and a relation R on Y we can define a relation r on X by saying that $x\ r\ y$ if and only if $f(x)\ R\ f(y)$. This is especially interesting when f is a bijection as all reasonable properties of R are inherited by r. This section treats mostly the case when R is an order relation and f is a bijection. The standard Isabelle's `Order` theory defines the notion of a space of order isomorphisms between two sets relative to a relation. We expand that material proving that order isomrphisms preserve interesting properties of the relation.

We call the relation created by a relation on Y and a mapping $f : X \to Y$ the `InducedRelation(f,R)`.

```
definition
  InducedRelation(f,R) ≡
  {p ∈ domain(f)×domain(f). ⟨f(fst(p)),f(snd(p))⟩ ∈ R}
```

A reformulation of the definition of the relation induced by a function.

```
lemma def_of_ind_relA:
  assumes ⟨x,y⟩ ∈ InducedRelation(f,R)
  shows ⟨f(x),f(y)⟩ ∈ R
  using assms InducedRelation_def by simp
```

A reformulation of the definition of the relation induced by a function, kind of converse of `def_of_ind_relA`.

```
lemma def_of_ind_relB: assumes f:A→B and
  x∈A  y∈A and ⟨f(x),f(y)⟩ ∈ R
  shows ⟨x,y⟩ ∈ InducedRelation(f,R)
  using assms func1_1_L1 InducedRelation_def by simp
```

A property of order isomorphisms that is missing from standard Isabelle's `Order.thy`.

```
lemma ord_iso_apply_conv:
  assumes f ∈ ord_iso(A,r,B,R) and
  ⟨f(x),f(y)⟩ ∈ R and x∈A  y∈A
  shows ⟨x,y⟩ ∈ r
  using assms ord_iso_def by simp
```

The next lemma tells us where the induced relation is defined

```
lemma ind_rel_domain:
  assumes  R ⊆ B×B and f:A→B
  shows InducedRelation(f,R) ⊆ A×A
  using assms func1_1_L1 InducedRelation_def
  by auto
```

A bijection is an order homomorphisms between a relation and the induced one.

```
lemma bij_is_ord_iso: assumes A1: f ∈ bij(A,B)
  shows f ∈ ord_iso(A,InducedRelation(f,R),B,R)
proof -
  let r = InducedRelation(f,R)
  { fix x y assume A2: x∈A  y∈A
    have ⟨x,y⟩ ∈ r ⟷ ⟨f(x),f(y)⟩ ∈ R
    proof
      assume ⟨x,y⟩ ∈ r then show ⟨f(x),f(y)⟩ ∈ R
 using def_of_ind_relA by simp
    next assume ⟨f(x),f(y)⟩ ∈ R
      with A1 A2 show ⟨x,y⟩ ∈ r
 using bij_is_fun def_of_ind_relB by blast
    qed }
  with A1 show f ∈ ord_iso(A,InducedRelation(f,R),B,R)
    using ord_isoI by simp
qed
```

An order isomoprhism preserves antisymmetry.

```
lemma ord_iso_pres_antsym: assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: antisym(R)
  shows antisym(r)
proof -
  { fix x y
    assume A4: ⟨x,y⟩ ∈ r   ⟨y,x⟩ ∈ r
    from A1 have f ∈ inj(A,B)
      using ord_iso_is_bij bij_is_inj by simp
    moreover
    from A1 A2 A4 have
      ⟨f(x), f(y)⟩ ∈ R and ⟨f(y), f(x)⟩ ∈ R
      using ord_iso_apply by auto
    with A3 have f(x) = f(y) by (rule Fol1_L4)
    moreover from A2 A4 have x∈A  y∈A by auto
    ultimately have x=y by (rule inj_apply_equality)
  } then have ∀x y. ⟨x,y⟩ ∈ r ∧ ⟨y,x⟩ ∈ r ⟶ x=y by auto
  then show antisym(r) using imp_conj antisym_def
    by simp
qed
```

Order isomoprhisms preserve transitivity.

```
lemma ord_iso_pres_trans: assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: trans(R)
  shows trans(r)
proof -
  { fix x y z
    assume A4: ⟨x, y⟩ ∈ r   ⟨y, z⟩ ∈ r
    note A1
    moreover
    from A1 A2 A4 have
      ⟨f(x), f(y)⟩ ∈ R ∧ ⟨f(y), f(z)⟩ ∈ R
```

```
      using ord_iso_apply by auto
    with A3 have ⟨f(x),f(z)⟩ ∈ R by (rule Fol1_L3)
    moreover from A2 A4 have x∈A  z∈A by auto
    ultimately have ⟨x, z⟩ ∈ r using ord_iso_apply_conv
      by simp
  } then have  ∀ x y z. ⟨x, y⟩ ∈ r ∧ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
    by blast
  then show trans(r) by (rule Fol1_L2)
qed
```

Order isomorphisms preserve totality.

```
lemma ord_iso_pres_tot: assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: R  {is total on} B
  shows r  {is total on} A
proof -
  { fix x y
    assume x∈A  y∈A  ⟨x,y⟩ ∉ r
    with A1 have ⟨f(x),f(y)⟩ ∉ R using ord_iso_apply_conv
      by auto
    moreover
    from A1 have f:A→B using ord_iso_is_bij bij_is_fun
      by simp
    with A3 ‹x∈A›  ‹y∈A› have
      ⟨f(x),f(y)⟩ ∈  R ∨ ⟨f(y),f(x)⟩ ∈  R
      using apply_funtype IsTotal_def by simp
    ultimately have ⟨f(y),f(x)⟩ ∈  R by simp
    with A1 ‹x∈A›  ‹y∈A› have ⟨y,x⟩ ∈ r
      using ord_iso_apply_conv  by simp
  } then have ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ r ∨  ⟨y,x⟩ ∈ r
    by blast
  then show r  {is total on} A using IsTotal_def
    by simp
qed
```

Order isomorphisms preserve linearity.

```
lemma ord_iso_pres_lin: assumes f ∈ ord_iso(A,r,B,R) and
  r ⊆ A×A and IsLinOrder(B,R)
  shows IsLinOrder(A,r)
  using assms ord_iso_pres_antsym ord_iso_pres_trans ord_iso_pres_tot
    IsLinOrder_def by simp
```

If a relation is a linear order, then the relation induced on another set by a bijection is also a linear order.

```
lemma ind_rel_pres_lin:
  assumes A1: f ∈ bij(A,B) and A2: IsLinOrder(B,R)
  shows IsLinOrder(A,InducedRelation(f,R))
proof -
  let r = InducedRelation(f,R)
  from A1 have f ∈ ord_iso(A,r,B,R) and r ⊆ A×A
```

```
    using bij_is_ord_iso domain_of_bij InducedRelation_def
    by auto
  with A2 show IsLinOrder(A,r) using ord_iso_pres_lin
    by simp
qed
```

The image by an order isomorphism of a bounded above and nonempty set is bounded above.

```
lemma ord_iso_pres_bound_above:
  assumes A1: f ∈ ord_iso(A,r,B,R) and A2: r ⊆ A×A and
  A3: IsBoundedAbove(C,r)   C≠0
  shows IsBoundedAbove(f(C),R)   f(C) ≠ 0
proof -
  from A3 obtain u where I: ∀x∈C. ⟨x,u⟩ ∈ r
    using IsBoundedAbove_def by auto
  from A1 have f:A→B using ord_iso_is_bij bij_is_fun
    by simp
  from A2 A3 have C⊆A using Order_ZF_3_L1A by blast
  from A3 obtain x where x∈C by auto
  with A2 I have u∈A by auto
  { fix y assume y ∈ f(C)
    with ‹f:A→B› ‹C⊆A› obtain x where x∈C and y = f(x)
      using func_imagedef by auto
    with A1 I ‹C⊆A›  ‹u∈A› have ⟨y,f(u)⟩ ∈ R
      using ord_iso_apply by auto
  } then have ∀y ∈ f(C).  ⟨y,f(u)⟩ ∈ R by simp
  then show IsBoundedAbove(f(C),R) by (rule Order_ZF_3_L10)
  from A3 ‹f:A→B› ‹C⊆A› show f(C) ≠ 0 using func1_1_L15A
    by simp
qed
```

Order isomorphisms preserve the property of having a minimum.

```
lemma ord_iso_pres_has_min:
  assumes A1: f ∈ ord_iso(A,r,B,R) and  A2: r ⊆ A×A and
  A3: C⊆A and A4: HasAminimum(R,f(C))
  shows HasAminimum(r,C)
proof -
  from A4 obtain m where
    I: m ∈ f(C) and II: ∀y ∈ f(C). ⟨m,y⟩ ∈ R
    using HasAminimum_def by auto
  let k = converse(f)(m)
  from A1 have f:A→B using ord_iso_is_bij bij_is_fun
    by simp
  from A1 have f ∈ inj(A,B) using ord_iso_is_bij bij_is_inj
    by simp
  with A3 I have k ∈ C and III: f(k) = m
    using inj_inv_back_in_set by auto
  moreover
  { fix x assume A5: x∈C
```

```
    with A3 II ‹f:A→B› ‹k ∈ C› III have
      k ∈ A   x∈A  ⟨f(k),f(x)⟩ ∈ R
      using func_imagedef by auto
    with A1 have ⟨k,x⟩ ∈ r using ord_iso_apply_conv
      by simp
  } then have ∀x∈C.  ⟨k,x⟩ ∈ r by simp
  ultimately show HasAminimum(r,C) using HasAminimum_def by auto
qed
```

Order isomorhisms preserve the images of relations. In other words taking the image of a point by a relation commutes with the function.

```
lemma ord_iso_pres_rel_image:
  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and
  A3: a∈A
  shows f(r{a}) = R{f(a)}
proof
  from A1 have f:A→B using ord_iso_is_bij bij_is_fun
    by simp
  moreover from A2 A3 have I: r{a} ⊆ A by auto
  ultimately have I: f(r{a}) = {f(x). x ∈ r{a} }
    using func_imagedef by simp
  { fix y assume A4: y ∈ f(r{a})
    with I obtain x where
      x ∈ r{a} and II: y = f(x)
      by auto
    with A1 A2 have ⟨f(a),f(x)⟩ ∈ R using ord_iso_apply
      by auto
    with II have y ∈  R{f(a)} by auto
  } then show  f(r{a}) ⊆ R{f(a)} by auto
  { fix y assume A5: y ∈ R{f(a)}
    let x = converse(f)(y)
    from A2 A5 have
      ⟨f(a),y⟩ ∈ R  f(a) ∈ B  and IV: y∈B
      by auto
    with A1 have III: ⟨converse(f)(f(a)),x⟩ ∈ r
      using ord_iso_converse by simp
    moreover from A1 A3 have converse(f)(f(a)) = a
      using ord_iso_is_bij left_inverse_bij by blast
    ultimately have f(x) ∈ {f(x). x ∈  r{a} }
      by auto
    moreover from A1 IV have f(x) = y
      using ord_iso_is_bij right_inverse_bij by blast
    moreover from A1 I have f(r{a}) = {f(x). x ∈  r{a} }
      using ord_iso_is_bij bij_is_fun func_imagedef by blast
    ultimately have y ∈ f(r{a}) by simp
  } then show R{f(a)} ⊆ f(r{a}) by auto
qed
```

Order isomorphisms preserve collections of upper bounds.

```
lemma ord_iso_pres_up_bounds:
  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and
  A3: C⊆A
  shows {f(r{a}). a∈C} = {R{b}. b ∈ f(C)}
proof
  from A1 have f:A→B
      using ord_iso_is_bij bij_is_fun by simp
  { fix Y assume Y ∈ {f(r{a}). a∈C}
    then obtain a where a∈C and I: Y = f(r{a})
      by auto
    from A3 ‹a∈C› have a∈A by auto
    with A1 A2 have f(r{a}) = R{f(a)}
      using ord_iso_pres_rel_image by simp
    moreover from A3 ‹f:A→B› ‹a∈C› have f(a) ∈ f(C)
      using func_imagedef by auto
    ultimately have f(r{a}) ∈ { R{b}. b ∈ f(C) }
      by auto
    with I have Y ∈ { R{b}. b ∈ f(C) } by simp
  } then show {f(r{a}). a∈C} ⊆ {R{b}. b ∈ f(C)}
    by blast
  { fix Y assume Y ∈ {R{b}. b ∈ f(C)}
    then obtain b where b ∈ f(C) and II: Y = R{b}
      by auto
    with A3 ‹f:A→B› obtain a where a∈C and b = f(a)
      using func_imagedef by auto
    with A3 II have a∈A and Y = R{f(a)} by auto
    with A1 A2 have Y = f(r{a})
      using ord_iso_pres_rel_image by simp
    with ‹a∈C› have Y ∈ {f(r{a}). a∈C} by auto
  } then show {R{b}. b ∈ f(C)} ⊆ {f(r{a}). a∈C}
    by auto
qed
```

The image of the set of upper bounds is the set of upper bounds of the image.

```
lemma ord_iso_pres_min_up_bounds:
  assumes A1: f ∈ ord_iso(A,r,B,R) and  A2: r ⊆ A×A  R ⊆ B×B and
  A3: C⊆A and A4: C≠0
  shows f(⋂a∈C. r{a}) = (⋂b∈f(C). R{b})
proof -
  from A1 have f ∈ inj(A,B)
    using ord_iso_is_bij bij_is_inj by simp
  moreover note A4
  moreover from A2 A3 have ∀a∈C. r{a} ⊆ A by auto
  ultimately have
    f(⋂a∈C. r{a}) = ( ⋂a∈C. f(r{a}) )
    using inj_image_of_Inter by simp
```

```
  also from A1 A2 A3 have
    ( ⋂a∈C. f(r{a}) ) = ( ⋂b∈f(C). R{b} )
    using ord_iso_pres_up_bounds by simp
  finally show f(⋂a∈C. r{a}) = (⋂b∈f(C). R{b})
    by simp
qed
```

Order isomorphisms preserve completeness.

```
lemma ord_iso_pres_compl:
  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and A3: R {is complete}
  shows r {is complete}
proof -
  { fix C
    assume A4: IsBoundedAbove(C,r)  C≠0
    with A1 A2 A3 have
      HasAminimum(R,⋂b ∈ f(C). R{b})
      using ord_iso_pres_bound_above IsComplete_def
      by simp
    moreover
    from A2 ‹IsBoundedAbove(C,r)› have I: C ⊆ A using Order_ZF_3_L1A
      by blast
    with A1 A2 ‹C≠0› have f(⋂a∈C. r{a}) = (⋂b∈f(C). R{b})
      using ord_iso_pres_min_up_bounds by simp
    ultimately have HasAminimum(R,f(⋂a∈C. r{a}))
      by simp
    moreover
    from A2 have ∀a∈C. r{a} ⊆ A
      by auto
    then have ( ⋂a∈C. r{a} ) ⊆ A using inter_subsets_subset
      by simp
    moreover note A1 A2
    ultimately have HasAminimum(r, ⋂a∈C. r{a} )
      using ord_iso_pres_has_min by simp
  } then show r {is complete} using IsComplete_def
    by simp
qed
```

If the original relation is complete, then the induced one is complete.

```
lemma ind_rel_pres_compl: assumes A1: f ∈ bij(A,B)
  and A2: R ⊆ B×B and A3: R {is complete}
  shows InducedRelation(f,R) {is complete}
proof -
  let r = InducedRelation(f,R)
  from A1 have f ∈ ord_iso(A,r,B,R)
    using bij_is_ord_iso by simp
  moreover from A1 A2 have r ⊆ A×A
    using bij_is_fun ind_rel_domain by simp
  moreover note A2 A3
```

```
  ultimately show r {is complete}
    using ord_iso_pres_compl by simp
qed


end
```

12 Semilattices and Lattices

```
theory Lattice_ZF imports Order_ZF_1a func1

begin
```

Lattices can be introduced in algebraic way as commutative idempotent ($x \cdot x = x$) semigroups or as partial orders with some additional properties. These two approaches are equivalent. In this theory we will use the order-theoretic approach.

12.1 Semilattices

We start with a relation r which is a partial order on a set L. Such situation is defined in `Order_ZF` as the predicate `IsPartOrder(L,r)`.

A partially ordered (L, r) set is a join-semilattice if each two-element subset of L has a supremum (i.e. the least upper bound).

```
definition
  IsJoinSemilattice(L,r) ≡
    r⊆L×L ∧ IsPartOrder(L,r) ∧ (∀x∈L. ∀y∈L. HasAsupremum(r,{x,y}))
```

A partially ordered (L, r) set is a meet-semilattice if each two-element subset of L has an infimum (i.e. the greatest lower bound).

```
definition
  IsMeetSemilattice(L,r) ≡
    r⊆L×L ∧ IsPartOrder(L,r) ∧ (∀x∈L. ∀y∈L. HasAnInfimum(r,{x,y}))
```

A partially ordered (L, r) set is a lattice if it is both join and meet-semilattice, i.e. if every two element set has a supremum (least upper bound) and infimum (greatest lower bound).

```
definition
  IsAlattice (infixl {is a lattice on} 90) where
  r {is a lattice on} L ≡ IsJoinSemilattice(L,r) ∧ IsMeetSemilattice(L,r)
```

Join is a binary operation whose value on a pair $\langle x, y\rangle$ is defined as the supremum of the set $\{x, y\}$.

```
definition
  Join(L,r) ≡ {⟨p,Supremum(r,{fst(p),snd(p)})⟩ . p ∈ L×L}
```

Meet is a binary operation whose value on a pair $\langle x,y\rangle$ is defined as the infimum of the set $\{x,y\}$.

definition
```
  Meet(L,r) ≡ {⟨p,Infimum(r,{fst(p),snd(p)})⟩ . p ∈ L×L}
```

Linear order is a lattice.

```
lemma lin_is_latt: assumes r⊆L×L and IsLinOrder(L,r)
  shows r {is a lattice on} L
proof -
  from assms(2) have IsPartOrder(L,r) using Order_ZF_1_L2 by simp
  with assms have IsMeetSemilattice(L,r) unfolding IsLinOrder_def IsMeetSemilattice_def
    using inf_sup_two_el(1) by auto
  moreover from assms ‹IsPartOrder(L,r)› have IsJoinSemilattice(L,r)
    unfolding IsLinOrder_def IsJoinSemilattice_def using inf_sup_two_el(3)
by auto
  ultimately show thesis unfolding IsAlattice_def by simp
qed
```

In a join-semilattice join is indeed a binary operation.

```
lemma join_is_binop: assumes IsJoinSemilattice(L,r)
  shows Join(L,r) :  L×L → L
proof -
  from assms have ∀p ∈ L×L. Supremum(r,{fst(p),snd(p)}) ∈ L
    unfolding IsJoinSemilattice_def IsPartOrder_def HasAsupremum_def us-
ing sup_in_space
    by auto
  then show thesis unfolding Join_def using ZF_fun_from_total by simp
qed
```

The value of `Join(L,r)` on a pair $\langle x,y\rangle$ is the supremum of the set $\{x,y\}$, hence its is greater or equal than both.

```
lemma join_val:
  assumes IsJoinSemilattice(L,r) x∈L y∈L
  defines j ≡ Join(L,r)⟨x,y⟩
  shows j∈L j = Supremum(r,{x,y}) ⟨x,j⟩ ∈ r ⟨y,j⟩ ∈ r
proof -
  from assms(1) have Join(L,r) :  L×L → L using join_is_binop by simp
  with assms(2,3,4) show j = Supremum(r,{x,y}) unfolding Join_def us-
ing ZF_fun_from_tot_val
    by auto
  from assms(2,3,4) ‹Join(L,r) : L×L → L›  show j∈L using apply_funtype
by simp
  from assms(1,2,3) have r ⊆ L×L antisym(r) HasAminimum(r,⋂z∈{x,y}.
r{z})
    unfolding IsJoinSemilattice_def IsPartOrder_def HasAsupremum_def by
auto
  with ‹j = Supremum(r,{x,y})› show ⟨x,j⟩ ∈ r and ⟨y,j⟩ ∈ r
    using sup_in_space(2) by auto
```

qed

In a meet-semilattice meet is indeed a binary operation.

```
lemma meet_is_binop: assumes IsMeetSemilattice(L,r)
  shows Meet(L,r) :  L×L → L
proof -
  from assms have ∀p ∈ L×L. Infimum(r,{fst(p),snd(p)}) ∈ L
    unfolding IsMeetSemilattice_def IsPartOrder_def HasAnInfimum_def us-
ing inf_in_space
    by auto
  then show thesis unfolding Meet_def using ZF_fun_from_total by simp
qed
```

The value of Meet(L,r) on a pair $\langle x, y\rangle$ is the infimum of the set $\{x, y\}$, hence is less or equal than both.

```
lemma meet_val:
  assumes IsMeetSemilattice(L,r) x∈L y∈L
  defines m ≡ Meet(L,r)⟨x,y⟩
  shows m∈L m = Infimum(r,{x,y}) ⟨m,x⟩ ∈ r ⟨m,y⟩ ∈ r
proof -
  from assms(1) have Meet(L,r) : L×L → L using meet_is_binop by simp
  with assms(2,3,4) show  m = Infimum(r,{x,y}) unfolding Meet_def us-
ing ZF_fun_from_tot_val
    by auto
  from assms(2,3,4) ‹Meet(L,r) : L×L → L›  show m∈L using apply_funtype
by simp
  from assms(1,2,3) have r ⊆ L×L antisym(r) HasAmaximum(r,⋂z∈{x,y}.
r-{z})
    unfolding IsMeetSemilattice_def IsPartOrder_def HasAnInfimum_def by
auto
  with ‹m = Infimum(r,{x,y})› show ⟨m,x⟩ ∈ r and ⟨m,y⟩ ∈ r
    using inf_in_space(2) by auto
qed
```

In a (nonempty) meet semi-lattice the relation down-directs the set.

```
lemma meet_down_directs: assumes IsMeetSemilattice(L,r) L≠0
  shows r {down-directs} L
proof -
  { fix x y assume x∈L y∈L
    let m = Meet(L,r)⟨x,y⟩
    from assms(1) ‹x∈L› ‹y∈L› have  m∈L ⟨m,x⟩ ∈ r ⟨m,y⟩ ∈ r
      using meet_val by auto
  } hence ∀x∈L. ∀y∈L. ∃m∈L. ⟨m,x⟩ ∈ r ∧⟨m,y⟩ ∈ r
    by blast
  with assms(2) show thesis unfolding DownDirects_def by simp
qed
```

In a (nonempty) join semi-lattice the relation up-directs the set.

```
lemma join_up_directs: assumes IsJoinSemilattice(L,r) L≠0
```

```
  shows r {up-directs} L
proof -
  { fix x y assume x∈L y∈L
    let m = Join(L,r)⟨x,y⟩
    from assms(1) ‹x∈L› ‹y∈L› have  m∈L ⟨x,m⟩ ∈ r ⟨y,m⟩ ∈ r
      using join_val by auto
  } hence ∀x∈L. ∀y∈L. ∃m∈L. ⟨x,m⟩ ∈ r ∧⟨y,m⟩ ∈ r
    by blast
  with assms(2) show thesis unfolding UpDirects_def by simp
qed
```

The next locale defines a a notation for join-semilattice. We will use the ⊔ symbol rather than more common ∨ to avoid confusion with logical "or".

```
locale join_semilatt =
  fixes L
  fixes r
  assumes joinLatt: IsJoinSemilattice(L,r)
  fixes join (infixl ⊔ 71)
  defines join_def [simp]: x ⊔ y ≡ Join(L,r)⟨x,y⟩
  fixes sup (sup _ )
  defines sup_def [simp]: sup A  ≡ Supremum(r,A)
```

Join of the elements of the lattice is in the lattice.

```
lemma (in join_semilatt) join_props: assumes x∈L y∈L
  shows x⊔y ∈ L and x⊔y = sup {x,y}
proof -
  from joinLatt assms have Join(L,r)⟨x,y⟩ ∈ L using join_is_binop apply_funtype

    by blast
  thus x⊔y ∈ L by simp
  from joinLatt assms have Join(L,r)⟨x,y⟩ = Supremum(r,{x,y}) using join_val(2)

    by simp
  thus x⊔y = sup {x,y} by simp
qed
```

Join is associative.

```
lemma (in join_semilatt) join_assoc: assumes x∈L y∈L z∈L
  shows x⊔(y⊔z) = x⊔y⊔z
proof -
  from joinLatt assms(2,3) have x⊔(y⊔z) = x⊔(sup {y,z}) using join_val(2)
by simp
  also from assms joinLatt have ... = sup {sup {x}, sup {y,z}}
    unfolding IsJoinSemilattice_def IsPartOrder_def using join_props sup_inf_singl(2)

    by auto
  also have ... = sup {x,y,z}
  proof -
    let 𝒯 = {{x},{y,z}}
```

```
    from joinLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsJoinSemilattice_def IsPartOrder_def by auto
    moreover from joinLatt assms have ∀T∈𝒯. HasAsupremum(r,T)
      unfolding IsJoinSemilattice_def IsPartOrder_def using sup_inf_singl(1)
by blast
    moreover from joinLatt assms have HasAsupremum(r,{Supremum(r,T).T∈𝒯})
      unfolding IsJoinSemilattice_def IsPartOrder_def  HasAsupremum_def

      using sup_in_space(1) sup_inf_singl(2) by auto
    ultimately have Supremum(r,{Supremum(r,T).T∈𝒯}) = Supremum(r,⋃𝒯)
by (rule sup_sup)
    moreover have {Supremum(r,T).T∈𝒯} = {sup {x}, sup {y,z}} and ⋃𝒯
= {x,y,z}
      by auto
    ultimately show (sup {sup {x}, sup {y,z}}) =  sup {x,y,z} by simp
  qed
  also have ... = sup {sup {x,y}, sup {z}}
  proof -
    let 𝒯 = {{x,y},{z}}
    from joinLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsJoinSemilattice_def IsPartOrder_def by auto
    moreover from joinLatt assms have ∀T∈𝒯. HasAsupremum(r,T)
      unfolding IsJoinSemilattice_def IsPartOrder_def using sup_inf_singl(1)
by blast
    moreover from joinLatt assms have HasAsupremum(r,{Supremum(r,T).T∈𝒯})
      unfolding IsJoinSemilattice_def IsPartOrder_def  HasAsupremum_def

      using sup_in_space(1) sup_inf_singl(2) by auto
    ultimately have Supremum(r,{Supremum(r,T).T∈𝒯}) = Supremum(r,⋃𝒯)
by (rule sup_sup)
    moreover have {Supremum(r,T).T∈𝒯} = {sup {x,y}, sup {z}} and ⋃𝒯
= {x,y,z}
      by auto
    ultimately show (sup {x,y,z}) = sup {sup {x,y}, sup {z}} by auto
  qed
  also from assms joinLatt have ... = sup {sup {x,y}, z}
    unfolding IsJoinSemilattice_def IsPartOrder_def using join_props sup_inf_singl(2)

    by auto
  also from assms joinLatt have ... = (sup {x,y}) ⊔ z
    unfolding IsJoinSemilattice_def IsPartOrder_def using join_props by
auto
  also from joinLatt assms(1,2) have ... = x⊔y⊔z using join_val(2) by
simp
  finally show x⊔(y⊔z) = x⊔y⊔z by simp
qed
```

Join is idempotent.

```
lemma (in join_semilatt) join_idempotent: assumes x∈L shows x⊔x = x
```

```
  using joinLatt assms join_val(2) IsJoinSemilattice_def IsPartOrder_def
sup_inf_singl(2)
  by auto
```

The meet_semilatt locale is the dual of the join-semilattice locale defined above. We will use the ⊓ symbol to denote join, giving it ab bit higher precedence.

```
locale meet_semilatt =
  fixes L
  fixes r
  assumes meetLatt: IsMeetSemilattice(L,r)
  fixes join (infixl ⊓ 72)
  defines join_def [simp]: x ⊓ y ≡ Meet(L,r)⟨x,y⟩
  fixes sup (inf _ )
  defines sup_def [simp]: inf A  ≡ Infimum(r,A)
```

Meet of the elements of the lattice is in the lattice.

```
lemma (in meet_semilatt) meet_props: assumes x∈L y∈L
  shows x⊓y ∈ L and x⊓y = inf {x,y}
proof -
  from meetLatt assms have Meet(L,r)⟨x,y⟩ ∈ L using meet_is_binop apply_funtype

    by blast
  thus x⊓y ∈ L by simp
  from meetLatt assms have Meet(L,r)⟨x,y⟩ = Infimum(r,{x,y}) using meet_val(2)
by blast
  thus x⊓y = inf {x,y} by simp
qed
```

Meet is associative.

```
lemma (in meet_semilatt) meet_assoc: assumes x∈L y∈L z∈L
  shows x⊓(y⊓z) = x⊓y⊓z
proof -
  from meetLatt assms(2,3) have x⊓(y⊓z) = x⊓(inf {y,z}) using meet_val
by simp
  also from assms meetLatt have ... = inf {inf {x}, inf {y,z}}
    unfolding IsMeetSemilattice_def IsPartOrder_def using meet_props sup_inf_singl(4)

    by auto
  also have ... = inf {x,y,z}
  proof -
    let 𝒯 = {{x},{y,z}}
    from meetLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsMeetSemilattice_def IsPartOrder_def by auto
    moreover from meetLatt assms have ∀T∈𝒯. HasAnInfimum(r,T)
      unfolding IsMeetSemilattice_def IsPartOrder_def using sup_inf_singl(3)
by blast
```

```
    moreover from meetLatt assms have HasAnInfimum(r,{Infimum(r,T).T∈𝒯})
      unfolding IsMeetSemilattice_def IsPartOrder_def  HasAnInfimum_def

      using inf_in_space(1) sup_inf_singl(4) by auto
    ultimately have Infimum(r,{Infimum(r,T).T∈𝒯}) = Infimum(r,⋃𝒯) by
(rule inf_inf)
    moreover have {Infimum(r,T).T∈𝒯} = {inf {x}, inf {y,z}} and ⋃𝒯
= {x,y,z}
      by auto
    ultimately show (inf {inf {x}, inf {y,z}}) =  inf {x,y,z} by simp
  qed
  also have ... = inf {inf {x,y}, inf {z}}
  proof -
    let 𝒯 = {{x,y},{z}}
    from meetLatt have r ⊆ L×L antisym(r) trans(r)
      unfolding IsMeetSemilattice_def IsPartOrder_def by auto
    moreover from meetLatt assms have ∀T∈𝒯. HasAnInfimum(r,T)
      unfolding IsMeetSemilattice_def IsPartOrder_def using sup_inf_singl(3)
by blast
    moreover from meetLatt assms have HasAnInfimum(r,{Infimum(r,T).T∈𝒯})
      unfolding IsMeetSemilattice_def IsPartOrder_def HasAnInfimum_def

      using inf_in_space(1) sup_inf_singl(4) by auto
    ultimately have Infimum(r,{Infimum(r,T).T∈𝒯}) = Infimum(r,⋃𝒯) by
(rule inf_inf)
    moreover have {Infimum(r,T).T∈𝒯} = {inf {x,y}, inf {z}} and ⋃𝒯
= {x,y,z}
      by auto
    ultimately show (inf {x,y,z}) = inf {inf {x,y}, inf {z}} by auto
  qed
  also from assms meetLatt have ... = inf {inf {x,y}, z}
    unfolding IsMeetSemilattice_def IsPartOrder_def using meet_props sup_inf_singl(4)

    by auto
  also from assms meetLatt have ... = (inf {x,y}) ⊓ z
    unfolding IsMeetSemilattice_def IsPartOrder_def using meet_props by
auto
  also from meetLatt assms(1,2) have ... = x⊓y⊓z using meet_val by simp
  finally show x⊓(y⊓z) = x⊓y⊓z by simp
qed
```

Meet is idempotent.

```
lemma (in meet_semilatt) meet_idempotent: assumes x∈L shows x⊓x = x

  using meetLatt assms meet_val IsMeetSemilattice_def IsPartOrder_def
sup_inf_singl(4)
  by auto

end
```

13 Finite sets - introduction

theory `Finite_ZF` **imports** `ZF1 Nat_ZF_IML ZF.Cardinal func1`

begin

Standard Isabelle Finite.thy contains a very useful notion of finite powerset: the set of finite subsets of a given set. The definition, however, is specific to Isabelle and based on the notion of "datatype", obviously not something that belongs to ZF set theory. This theory file devolops the notion of finite powerset similarly as in Finite.thy, but based on standard library's Cardinal.thy. This theory file is intended to replace IsarMathLib's `Finite1` and `Finite_ZF_1` theories that are currently derived from the "datatype" approach.

13.1 Definition and basic properties of finite powerset

The goal of this section is to prove an induction theorem about finite powersets: if the empty set has some property and this property is preserved by adding a single element of a set, then this property is true for all finite subsets of this set.

We defined the finite powerset `FinPow(X)` as those elements of the powerset that are finite.

```
definition
  FinPow(X) ≡ {A ∈ Pow(X). Finite(A)}
```

The cardinality of an element of finite powerset is a natural number.

```
lemma card_fin_is_nat: assumes A ∈ FinPow(X)
  shows |A| ∈ nat and A ≈ |A|
  using assms FinPow_def Finite_def cardinal_cong nat_into_Card
    Card_cardinal_eq by auto
```

The cardinality of a finite set is a natural number.

```
lemma card_fin_is_nat1: assumes Finite(A) shows |A| ∈ nat
  using assms card_fin_is_nat(1) unfolding FinPow_def by auto
```

A reformulation of `card_fin_is_nat`: for a finit set A there is a bijection between $|A|$ and A.

```
lemma fin_bij_card: assumes A1: A ∈ FinPow(X)
  shows ∃b. b ∈ bij(|A|, A)
proof -
  from A1 have |A| ≈ A using card_fin_is_nat eqpoll_sym
    by blast
  then show thesis using eqpoll_def by auto
qed
```

If a set has the same number of elements as $n \in \mathbb{N}$, then its cardinality is n. Recall that in set theory a natural number n is a set that has n elements.

```
lemma card_card: assumes A ≈ n and n ∈ nat
  shows |A| = n
  using assms cardinal_cong nat_into_Card Card_cardinal_eq
  by auto
```

If we add a point to a finite set, the cardinality increases by one. To understand the second assertion $|A \cup \{a\}| = |A| \cup \{|A|\}$ recall that the cardinality $|A|$ of A is a natural number and for natural numbers we have $n+1 = n \cup \{n\}$.

```
lemma card_fin_add_one: assumes A1: A ∈ FinPow(X) and A2: a ∈ X-A
  shows
  |A ∪ {a}| = succ( |A| )
  |A ∪ {a}| = |A| ∪ {|A|}
proof -
  from A1 A2 have cons(a,A) ≈ cons( |A|, |A| )
    using card_fin_is_nat mem_not_refl cons_eqpoll_cong
    by auto
  moreover have  cons(a,A) = A ∪ {a} by (rule consdef)
  moreover have cons( |A|, |A| ) = |A| ∪ {|A|}
    by (rule consdef)
  ultimately have A∪{a} ≈ succ( |A| ) using succ_explained
    by simp
  with A1 show
    |A ∪ {a}| = succ( |A| ) and |A ∪ {a}| = |A| ∪ {|A|}
    using card_fin_is_nat card_card by auto
qed
```

We can decompose the finite powerset into collection of sets of the same natural cardinalities.

```
lemma finpow_decomp:
  shows FinPow(X) = (⋃n ∈ nat. {A ∈ Pow(X). A ≈ n})
  using Finite_def FinPow_def by auto
```

Finite powerset is the union of sets of cardinality bounded by natural numbers.

```
lemma finpow_union_card_nat:
  shows FinPow(X) = (⋃n ∈ nat. {A ∈ Pow(X). A ≲ n})
proof -
  have FinPow(X) ⊆ (⋃n ∈ nat. {A ∈ Pow(X). A ≲ n})
    using finpow_decomp FinPow_def eqpoll_imp_lepoll
    by auto
  moreover have
    (⋃n ∈ nat. {A ∈ Pow(X). A ≲ n}) ⊆ FinPow(X)
    using lepoll_nat_imp_Finite FinPow_def by auto
  ultimately show thesis by auto
qed
```

A different form of `finpow_union_card_nat` (see above) - a subset that has not more elements than a given natural number is in the finite powerset.

```
lemma lepoll_nat_in_finpow:
  assumes n ∈ nat   A ⊆ X  A ≲ n
  shows A ∈ FinPow(X)
  using assms finpow_union_card_nat by auto
```

Natural numbers are finite subsets of the set of natural numbers.

```
lemma nat_finpow_nat: assumes n ∈ nat shows n ∈ FinPow(nat)
  using assms nat_into_Finite nat_subset_nat FinPow_def
  by simp
```

A finite subset is a finite subset of itself.

```
lemma fin_finpow_self: assumes A ∈ FinPow(X) shows A ∈ FinPow(A)
  using assms FinPow_def by auto
```

A set is finite iff it is in its finite powerset.

```
lemma fin_finpow_iff: shows Finite(A) ⟷ A ∈ FinPow(A)
  unfolding FinPow_def by simp
```

Induction for finite powerset. This is smilar to the standard Isabelle's `Fin_induct`.

```
theorem FinPow_induct: assumes A1: P(0) and
  A2: ∀A ∈ FinPow(X). P(A) ⟶ (∀a∈X. P(A ∪ {a})) and
  A3: B ∈ FinPow(X)
  shows P(B)
proof -
  { fix n assume n ∈ nat
    moreover from A1 have I: ∀B∈Pow(X). B ≲ 0 ⟶ P(B)
      using lepoll_0_is_0 by auto
    moreover have ∀ k ∈ nat.
      (∀B ∈ Pow(X). (B ≲ k ⟶ P(B))) ⟶
      (∀B ∈ Pow(X). (B ≲ succ(k) ⟶ P(B)))
    proof -
      { fix k assume A4: k ∈ nat
 assume A5: ∀ B ∈ Pow(X). (B ≲ k ⟶ P(B))
 fix B assume A6: B ∈ Pow(X)  B ≲ succ(k)
 have P(B)
 proof -
   have B = 0 ⟶ P(B)
   proof -
     { assume B = 0
       then have B ≲ 0 using lepoll_0_iff
  by simp
       with I A6 have P(B) by simp
     } thus B = 0 ⟶ P(B) by simp
   qed
   moreover have B≠0 ⟶ P(B)
```

```
    proof -
      { assume B ≠ 0
        then obtain a where II: a∈B by auto
        let A = B - {a}
        from A6 II have A ⊆ X and A ≲ k
  using Diff_sing_lepoll by auto
        with A4 A5 have A ∈ FinPow(X) and P(A)
  using lepoll_nat_in_finpow finpow_decomp
  by auto
        with A2 A6 II have  P(A ∪ {a})
  by auto
        moreover from II have A ∪ {a} = B
  by auto
        ultimately have P(B) by simp
      } thus B≠0 ⟶ P(B) by simp
    qed
    ultimately show P(B) by auto
 qed
      } thus thesis by blast
    qed
    ultimately have ∀B ∈ Pow(X). (B ≲ n ⟶ P(B))
      by (rule ind_on_nat)
  } then have ∀n ∈ nat. ∀B ∈ Pow(X). (B ≲ n ⟶ P(B))
    by auto
  with A3 show P(B) using finpow_union_card_nat
    by auto
qed
```

A subset of a finite subset is a finite subset.

```
lemma subset_finpow: assumes A ∈ FinPow(X) and B ⊆ A
  shows B ∈ FinPow(X)
  using assms FinPow_def subset_Finite by auto
```

If we subtract anything from a finite set, the resulting set is finite.

```
lemma diff_finpow:
  assumes A ∈ FinPow(X) shows A-B ∈ FinPow(X)
  using assms subset_finpow by blast
```

If we remove a point from a finites subset, we get a finite subset.

```
corollary fin_rem_point_fin: assumes A ∈ FinPow(X)
  shows A - {a} ∈ FinPow(X)
  using assms diff_finpow by simp
```

Cardinality of a nonempty finite set is a successsor of some natural number.

```
lemma card_non_empty_succ:
  assumes A1: A ∈ FinPow(X) and A2: A ≠ 0
  shows ∃n ∈ nat. |A| = succ(n)
proof -
```

```
  from A2 obtain a where a ∈ A by auto
  let B = A - {a}
  from A1 ‹a ∈ A› have
    B ∈ FinPow(X) and a ∈ X - B
    using FinPow_def fin_rem_point_fin by auto
  then have |B ∪ {a}| = succ( |B| )
    using card_fin_add_one by auto
  moreover from ‹a ∈ A› ‹B ∈ FinPow(X)› have
    A = B ∪ {a} and |B| ∈ nat
    using card_fin_is_nat by auto
  ultimately show ∃n ∈ nat. |A| = succ(n) by auto
qed
```

Nonempty set has non-zero cardinality. This is probably true without the assumption that the set is finite, but I couldn't derive it from standard Isabelle theorems.

```
lemma card_non_empty_non_zero:
  assumes A ∈ FinPow(X) and A ≠ 0
  shows |A| ≠ 0
proof -
  from assms obtain n where |A| = succ(n)
    using card_non_empty_succ by auto
  then show |A| ≠ 0 using succ_not_0
    by simp
qed
```

Another variation on the induction theme: If we can show something holds for the empty set and if it holds for all finite sets with at most k elements then it holds for all finite sets with at most $k + 1$ elements, the it holds for all finite sets.

```
theorem FinPow_card_ind: assumes A1: P(0) and
  A2: ∀k∈nat.
  (∀A ∈ FinPow(X). A ≲ k ⟶ P(A)) ⟶
  (∀A ∈ FinPow(X). A ≲ succ(k) ⟶ P(A))
  and A3: A ∈  FinPow(X) shows P(A)
proof -
  from A3 have |A| ∈ nat and A ∈  FinPow(X) and A ≲ |A|
    using card_fin_is_nat eqpoll_imp_lepoll by auto
  moreover have ∀n ∈ nat. (∀A ∈ FinPow(X).
    A ≲ n ⟶ P(A))
  proof
    fix n assume n ∈ nat
    moreover from A1 have ∀A ∈ FinPow(X). A ≲ 0 ⟶ P(A)
      using lepoll_0_is_0 by auto
    moreover note A2
    ultimately show
      ∀A ∈ FinPow(X). A ≲ n ⟶ P(A)
      by (rule ind_on_nat)
```

```
  qed
  ultimately show P(A) by simp
qed
```

Another type of induction (or, maybe recursion). In the induction step we try to find a point in the set that if we remove it, the fact that the property holds for the smaller set implies that the property holds for the whole set.

```
lemma FinPow_ind_rem_one: assumes A1: P(0) and
  A2: ∀ A ∈ FinPow(X). A ≠ 0 ⟶ (∃a∈A. P(A-{a}) ⟶ P(A))
  and A3: B ∈  FinPow(X)
  shows P(B)
proof -
  note A1
  moreover have ∀k∈nat.
  (∀B ∈ FinPow(X). B ≲ k ⟶ P(B)) ⟶
  (∀C ∈ FinPow(X). C ≲ succ(k) ⟶ P(C))
  proof -
    { fix k assume k ∈ nat
      assume A4: ∀B ∈ FinPow(X). B ≲ k ⟶ P(B)
      have ∀C ∈ FinPow(X). C ≲ succ(k) ⟶ P(C)
      proof -
 { fix C assume C ∈ FinPow(X)
   assume C ≲ succ(k)
   note A1
   moreover
   { assume C ≠ 0
     with A2 ‹C ∈ FinPow(X)› obtain a where
       a∈C and P(C-{a}) ⟶ P(C)
       by auto
     with A4 ‹C ∈ FinPow(X)› ‹C ≲ succ(k)›
     have P(C) using Diff_sing_lepoll fin_rem_point_fin
       by simp }
   ultimately have P(C) by auto
 } thus thesis by simp
      qed
    } thus thesis by blast
  qed
  moreover note A3
  ultimately show P(B) by (rule FinPow_card_ind)
qed
```

Yet another induction theorem. This is similar, but slightly more complicated than `FinPow_ind_rem_one`. The difference is in the treatment of the empty set to allow to show properties that are not true for empty set.

```
lemma FinPow_rem_ind: assumes A1: ∀A ∈ FinPow(X).
  A = 0 ∨ (∃a∈A. A = {a} ∨ P(A-{a}) ⟶ P(A))
  and A2: A ∈  FinPow(X) and A3: A≠0
  shows P(A)
proof -
```

```
  have 0 = 0 ∨ P(0) by simp
  moreover have
    ∀k∈nat.
    (∀B ∈ FinPow(X). B ≲ k ⟶ (B=0 ∨ P(B))) ⟶
    (∀A ∈ FinPow(X). A ≲ succ(k) ⟶ (A=0 ∨ P(A)))
  proof -
    { fix k assume k ∈ nat
      assume A4: ∀B ∈ FinPow(X). B ≲ k ⟶ (B=0 ∨ P(B))
      have ∀A ∈ FinPow(X). A ≲ succ(k) ⟶ (A=0 ∨ P(A))
      proof -
 { fix A assume A ∈ FinPow(X)
   assume A ≲ succ(k)  A≠0
   from A1 ‹A ∈ FinPow(X)› ‹A≠0› obtain a
     where a∈A and A = {a} ∨ P(A-{a}) ⟶ P(A)
     by auto
   let B = A-{a}
   from A4 ‹A ∈ FinPow(X)› ‹A ≲ succ(k)› ‹a∈A›
   have B = 0 ∨ P(B)
     using Diff_sing_lepoll fin_rem_point_fin
     by simp
   with ‹a∈A› ‹A = {a} ∨ P(A-{a}) ⟶ P(A)›
   have P(A) by auto
 } thus  thesis by auto
      qed
    } thus thesis by blast
  qed
  moreover note A2
  ultimately have A=0 ∨ P(A) by (rule FinPow_card_ind)
  with A3 show P(A) by simp
qed
```

If a family of sets is closed with respect to taking intersections of two sets then it is closed with respect to taking intersections of any nonempty finite collection.

```
lemma inter_two_inter_fin:
  assumes A1: ∀V∈T. ∀W∈T. V ∩ W ∈ T and
  A2: N ≠ 0 and A3: N ∈ FinPow(T)
  shows ⋂N ∈ T
proof -
  have 0 = 0 ∨ (⋂0 ∈ T) by simp
  moreover have ∀M ∈ FinPow(T). (M = 0 ∨ ⋂M ∈ T) ⟶
    (∀W ∈ T. M∪{W} = 0 ∨ ⋂(M ∪ {W}) ∈ T)
  proof -
    { fix M assume M ∈ FinPow(T)
      assume A4: M = 0 ∨ ⋂M ∈ T
      { assume M = 0
 hence ∀W ∈ T. M∪{W} = 0 ∨ ⋂(M ∪ {W}) ∈ T
   by auto }
      moreover
```

```
      { assume M ≠ 0
 with A4 have ⋂M ∈ T by simp
 { fix W assume W ∈ T
   from ‹M ≠ 0› have ⋂(M ∪ {W}) = (⋂M) ∩ W
     by auto
   with A1 ‹⋂M ∈ T› ‹W ∈ T› have ⋂(M ∪ {W}) ∈ T
     by simp
 } hence ∀W ∈ T. M∪{W} = 0 ∨ ⋂(M ∪ {W}) ∈ T
   by simp }
      ultimately have ∀W ∈ T. M∪{W} = 0 ∨ ⋂(M ∪ {W}) ∈ T
 by blast
    } thus thesis by simp
  qed
  moreover note ‹N ∈ FinPow(T)›
  ultimately have N = 0 ∨ (⋂N ∈ T)
    by (rule FinPow_induct)
  with A2 show (⋂N ∈ T) by simp
qed
```

If a family of sets contains the empty set and is closed with respect to taking unions of two sets then it is closed with respect to taking unions of any finite collection.

```
lemma union_two_union_fin:
  assumes A1: 0 ∈ C and A2: ∀A∈C. ∀B∈C. A∪B ∈ C and
  A3: N ∈ FinPow(C)
  shows ⋃N ∈ C
proof -
  from ‹0 ∈ C› have ⋃0 ∈ C by simp
  moreover have ∀M ∈ FinPow(C). ⋃M ∈ C ⟶ (∀A∈C. ⋃(M ∪ {A}) ∈ C)
  proof -
    { fix M assume M ∈ FinPow(C)
      assume ⋃M ∈ C
      fix A assume A∈C
      have ⋃(M ∪ {A}) = (⋃M) ∪ A by auto
      with A2 ‹⋃M ∈ C›  ‹A∈C› have  ⋃(M ∪ {A}) ∈ C
 by simp
    } thus thesis by simp
  qed
  moreover note ‹N ∈ FinPow(C)›
  ultimately show ⋃N ∈ C by (rule FinPow_induct)
qed
```

Empty set is in finite power set, hence finite power set is never empty.

```
lemma empty_in_finpow: shows ∅ ∈ FinPow(X) and FinPow(X)≠∅
  using FinPow_def by auto
```

Singleton is in the finite powerset.

```
lemma singleton_in_finpow: assumes x ∈ X
```

```
  shows {x} ∈ FinPow(X) using assms FinPow_def by simp
```

If a set is nonempty then its finite power set contains a nonempty set.

```
lemma finpow_nempty_nempty: assumes X≠∅ shows FinPow(X)\{∅} ≠ ∅
  using assms singleton_in_finpow by blast
```

Union of two finite subsets is a finite subset.

```
lemma union_finpow: assumes A ∈ FinPow(X) and B ∈ FinPow(X)
  shows A ∪ B ∈ FinPow(X)
  using assms FinPow_def by auto
```

Union of finite number of finite sets is finite.

```
lemma fin_union_finpow: assumes M ∈ FinPow(FinPow(X))
  shows ⋃M ∈ FinPow(X)
  using assms empty_in_finpow union_finpow union_two_union_fin
  by simp
```

If a set is finite after removing one element, then it is finite.

```
lemma rem_point_fin_fin:
  assumes A1: x ∈ X and A2: A - {x} ∈ FinPow(X)
  shows A ∈ FinPow(X)
proof -
  from assms have (A - {x}) ∪ {x} ∈ FinPow(X)
    using singleton_in_finpow union_finpow by simp
  moreover have A ⊆ (A - {x}) ∪ {x} by auto
  ultimately show A ∈ FinPow(X)
    using FinPow_def subset_Finite by auto
qed
```

An image of a finite set is finite.

```
lemma fin_image_fin: assumes ∀V∈B. K(V)∈C and N ∈ FinPow(B)
  shows {K(V). V∈N} ∈ FinPow(C)
proof -
  have {K(V). V∈0} ∈ FinPow(C) using FinPow_def
    by auto
  moreover have ∀A ∈ FinPow(B).
    {K(V). V∈A} ∈ FinPow(C) ⟶ (∀a∈B. {K(V). V ∈ (A ∪ {a})} ∈ FinPow(C))
  proof -
    { fix A assume A ∈ FinPow(B)
      assume  {K(V). V∈A} ∈ FinPow(C)
      fix a assume a∈B
      have  {K(V). V ∈ (A ∪ {a})} ∈ FinPow(C)
      proof -
 have {K(V). V ∈ (A ∪ {a})} = {K(V). V∈A} ∪ {K(a)}
   by auto
 moreover note ‹{K(V). V∈A} ∈ FinPow(C)›
 moreover from ‹∀V∈B. K(V) ∈ C›  ‹a∈B› have {K(a)} ∈  FinPow(C)
   using singleton_in_finpow by simp
```

```
 ultimately show thesis using union_finpow by simp
      qed
    } thus thesis by simp
  qed
  moreover note ‹N ∈ FinPow(B)›
  ultimately show {K(V). V∈N} ∈ FinPow(C)
    by (rule FinPow_induct)
qed
```

A variant of fin_image_fin with a bit weaker first assumption:

```
lemma fin_image_fin1: assumes ∀V∈N. K(V)∈C and N ∈ FinPow(B)
  shows {K(V). V∈N} ∈ FinPow(C)
proof -
  from assms(2) have N ∈ FinPow(N)
    using fin_finpow_self by simp
  with assms(1) show thesis by (rule fin_image_fin)
qed
```

An image of a nonempty finite set is a nonempty finite set.

```
lemma fin_image_fin0: assumes N ∈ FinPow(B)\{∅} and ∀V∈N. K(V)∈C
  shows  {K(V). V∈N} ∈ FinPow(C)\{∅}
  using assms fin_image_fin1 by auto
```

If a set X is finite then the set $\{K(x).x \in X\}$ is also finite. It's basically standard Isalelle/ZF Finite_RepFun in nicer notation.

```
lemma fin_rep_fin: assumes Finite(X) shows Finite({K(x). x∈X})
  using assms Finite_RepFun by simp
```

The image of a singleton by any function is finite. It's of course either empty or has exactly one element, but showing that it's a finite subset of the codomain is good enough for us.

```
lemma image_singleton_fin: assumes f:X→Y shows f{x} ∈ FinPow(Y)
proof -
  { assume x∈X
    with assms have f{x} ∈ FinPow(Y)
      using singleton_image singleton_in_finpow by simp
  }
  moreover
  { assume x∉X
    with assms have f{x} ∈ FinPow(Y)
      using arg_not_in_domain(1) empty_in_finpow by simp
  }
  ultimately show thesis by auto
qed
```

Union of a finite indexed family of finite sets is finite.

```
lemma union_fin_list_fin:
  assumes A1: n ∈ nat and A2: ∀k ∈ n. N(k) ∈ FinPow(X)
```

```
  shows
  {N(k). k ∈ n} ∈  FinPow(FinPow(X)) and (⋃k ∈ n. N(k)) ∈ FinPow(X)
proof -
  from A1 have n ∈ FinPow(n)
    using nat_finpow_nat fin_finpow_self by auto
  with A2 show {N(k). k ∈ n} ∈  FinPow(FinPow(X))
    by (rule fin_image_fin)
  then show (⋃k ∈ n. N(k)) ∈ FinPow(X)
    using fin_union_finpow by simp
qed

end
```

14 Finite sets

```
theory Finite1 imports ZF.EquivClass ZF.Finite func1 ZF1

begin
```

This theory extends Isabelle standard `Finite` theory. It is obsolete and should not be used for new development. Use the `Finite_ZF` instead.

14.1 Finite powerset

In this section we consider various properties of `Fin` datatype (even though there are no datatypes in ZF set theory).

In `Topology_ZF` theory we consider induced topology that is obtained by taking a subset of a topological space. To show that a topology restricted to a subset is also a topology on that subset we may need a fact that if T is a collection of sets and A is a set then every finite collection $\{V_i\}$ is of the form $V_i = U_i \cap A$, where $\{U_i\}$ is a finite subcollection of T. This is one of those trivial facts that require suprisingly long formal proof. Actually, the need for this fact is avoided by requiring intersection two open sets to be open (rather than intersection of a finite number of open sets). Still, the fact is left here as an example of a proof by induction. We will use `Fin_induct` lemma from Finite.thy. First we define a property of finite sets that we want to show.

```
definition
  Prfin(T,A,M) ≡ ( (M = 0) | (∃N∈ Fin(T). ∀V∈ M. ∃ U∈ N. (V = U∩A)))
```

Now we show the main induction step in a separate lemma. This will make the proof of the theorem FinRestr below look short and nice. The premises of the `ind_step` lemma are those needed by the main induction step in lemma `Fin_induct` (see standard Isabelle's Finite.thy).

```
lemma ind_step: assumes  A: ∀ V∈ TA. ∃ U∈T. V=U∩A
```

```
  and A1: W∈TA and A2: M∈ Fin(TA)
  and A3: W∉M and A4: Prfin(T,A,M)
  shows Prfin(T,A,cons(W,M))
proof -
  { assume A7: M=0 have Prfin(T, A, cons(W, M))
    proof-
      from A1 A obtain U where A5: U∈T and A6: W=U∩A by fast
      let N = {U}
      from A5 have T1: N ∈ Fin(T) by simp
      from A7 A6 have T2: ∀V∈ cons(W,M). ∃ U∈N. V=U∩A by simp
      from A7 T1 T2 show  Prfin(T, A, cons(W, M))
 using Prfin_def by auto
    qed }
  moreover
  { assume A8:M≠0 have Prfin(T, A, cons(W, M))
    proof-
      from A1 A obtain U where A5: U∈T and A6:W=U∩A by fast
      from A8 A4 obtain N0
 where A9: N0∈ Fin(T) and A10: ∀V∈ M. ∃ U0∈ N0. (V = U0∩A)
 using Prfin_def by auto
      let N = cons(U,N0)
      from A5 A9 have N ∈ Fin(T) by simp
      moreover from A10 A6 have ∀V∈ cons(W,M). ∃ U∈N. V=U∩A by simp
      ultimately have ∃ N∈ Fin(T).∀V∈ cons(W,M). ∃ U∈N. V=U∩A by auto
      with A8 show Prfin(T, A, cons(W, M))
 using Prfin_def by simp
    qed }
  ultimately show thesis by auto
qed
```

Now we are ready to prove the statement we need.

```
theorem FinRestr0: assumes A: ∀ V ∈ TA. ∃ U∈ T. V=U∩A
  shows ∀ M∈ Fin(TA). Prfin(T,A,M)
proof -
  { fix M
    assume M ∈ Fin(TA)
    moreover have Prfin(T,A,0) using Prfin_def by simp
    moreover
    { fix W M assume W∈TA M∈ Fin(TA) W∉M Prfin(T,A,M)
      with A have Prfin(T,A,cons(W,M)) by (rule ind_step) }
    ultimately have Prfin(T,A,M) by (rule Fin_induct)
  } thus thesis by simp
qed
```

This is a different form of the above theorem:

```
theorem ZF1FinRestr:
  assumes A1:M∈ Fin(TA) and A2: M≠0
  and A3: ∀ V∈ TA. ∃ U∈ T. V=U∩A
  shows ∃N∈ Fin(T). (∀V∈ M. ∃ U∈ N. (V = U∩A)) ∧ N≠0
```

```
proof -
  from A3 A1 have Prfin(T,A,M) using FinRestr0 by blast
  then have ∃N∈ Fin(T). ∀V∈ M. ∃ U∈ N. (V = U∩A)
    using A2 Prfin_def by simp
  then obtain N where
    D1:N∈ Fin(T) ∧ (∀V∈ M. ∃ U∈ N. (V = U∩A)) by auto
  with A2 have N≠0 by auto
  with D1 show thesis by auto
qed
```

Purely technical lemma used in Topology_ZF_1 to show that if a topology is T_2, then it is T_1.

```
lemma Finite1_L2:
  assumes A:∃U V. (U∈T ∧ V∈T ∧ x∈U ∧ y∈V ∧ U∩V=0)
  shows ∃U∈T. (x∈U ∧ y∉U)
proof -
  from A obtain U V where D1:U∈T ∧ V∈T ∧ x∈U ∧ y∈V ∧ U∩V=0 by auto
  with D1 show thesis by auto
qed
```

A collection closed with respect to taking a union of two sets is closed under taking finite unions. Proof by induction with the induction step formulated in a separate lemma.

```
lemma Finite1_L3_IndStep:
  assumes A1:∀A B. ((A∈C ∧ B∈C) ⟶ A∪B∈C)
  and A2: A∈C and A3: N∈Fin(C) and A4:A∉N and A5:⋃N ∈ C
  shows ⋃cons(A,N) ∈ C
proof -
  have ⋃ cons(A,N) = A∪ ⋃N by blast
  with A1 A2 A5 show thesis by simp
qed
```

The lemma: a collection closed with respect to taking a union of two sets is closed under taking finite unions.

```
lemma Finite1_L3:
  assumes A1: 0 ∈ C and A2: ∀A B. ((A∈C ∧ B∈C) ⟶ A∪B∈C) and
  A3: N∈ Fin(C)
  shows ⋃N∈C
proof -
  note A3
  moreover from A1 have ⋃0 ∈ C by simp
  moreover
  { fix A N
    assume A∈C N∈Fin(C) A∉N ⋃N ∈ C
    with A2 have ⋃cons(A,N) ∈ C by (rule Finite1_L3_IndStep) }
  ultimately show ⋃N∈ C by (rule Fin_induct)
qed
```

A collection closed with respect to taking a intersection of two sets is closed under taking finite intersections. Proof by induction with the induction step formulated in a separate lemma. This is sligltly more involved than the union case in `Finite1_L3`, because the intersection of empty collection is undefined (or should be treated as such). To simplify notation we define the property to be proven for finite sets as a separate notion.

```
definition
  IntPr(T,N) ≡ (N = 0 | ⋂N ∈ T)
```

The induction step.

```
lemma Finite1_L4_IndStep:
  assumes A1: ∀A B. ((A∈T ∧ B∈T) ⟶ A∩B∈T)
  and A2: A∈T and A3:N∈Fin(T) and A4:A∉N and A5:IntPr(T,N)
  shows IntPr(T,cons(A,N))
proof -
  { assume A6: N=0
    with A2 have IntPr(T,cons(A,N))
      using IntPr_def by simp }
  moreover
  { assume A7: N≠0 have IntPr(T, cons(A, N))
    proof -
      from A7 A5 A2 A1 have ⋂N ∩ A ∈ T using IntPr_def by simp
      moreover from A7 have ⋂cons(A, N) = ⋂N ∩ A by auto
      ultimately show IntPr(T, cons(A, N)) using IntPr_def by simp
    qed }
  ultimately show thesis by auto
qed
```

The lemma.

```
lemma Finite1_L4:
  assumes A1: ∀A B. A∈T ∧ B∈T ⟶ A∩B ∈ T
  and A2: N∈Fin(T)
  shows IntPr(T,N)
proof -
  note A2
  moreover have IntPr(T,0) using IntPr_def by simp
  moreover
  { fix A N
    assume A∈T N∈Fin(T) A∉N IntPr(T,N)
    with A1 have  IntPr(T,cons(A,N)) by (rule Finite1_L4_IndStep) }
  ultimately show IntPr(T,N) by (rule Fin_induct)
qed
```

Next is a restatement of the above lemma that does not depend on the IntPr meta-function.

```
lemma Finite1_L5:
  assumes A1: ∀A B. ((A∈T ∧ B∈T) ⟶ A∩B∈T)
  and A2: N≠0 and A3: N∈Fin(T)
  shows ⋂N ∈ T
```

```
proof -
  from A1 A3 have IntPr(T,N) using Finite1_L4 by simp
  with A2 show thesis using IntPr_def by simp
qed
```

The images of finite subsets by a meta-function are finite. For example in topology if we have a finite collection of sets, then closing each of them results in a finite collection of closed sets. This is a very useful lemma with many unexpected applications. The proof is by induction. The next lemma is the induction step.

```
lemma fin_image_fin_IndStep:
  assumes ∀V∈B. K(V)∈C
  and U∈B and N∈Fin(B) and U∉N and {K(V). V∈N}∈Fin(C)
  shows {K(V). V∈cons(U,N)} ∈ Fin(C)
  using assms by simp
```

The lemma:

```
lemma fin_image_fin:
  assumes A1: ∀V∈B. K(V)∈C and A2: N∈Fin(B)
  shows {K(V). V∈N} ∈ Fin(C)
proof -
  note A2
  moreover have {K(V). V∈0} ∈ Fin(C) by simp
  moreover
  { fix U N
    assume U∈B N∈Fin(B) U∉N {K(V). V∈N}∈Fin(C)
    with A1 have {K(V). V∈cons(U,N)} ∈ Fin(C)
      by (rule fin_image_fin_IndStep) }
  ultimately show thesis by (rule Fin_induct)
qed
```

The image of a finite set is finite.

```
lemma Finite1_L6A: assumes A1: f:X→Y and A2: N ∈ Fin(X)
  shows f(N) ∈ Fin(Y)
proof -
  from A1 have ∀x∈X. f(x) ∈ Y
    using apply_type by simp
  moreover note A2
  ultimately have {f(x). x∈N} ∈ Fin(Y)
    by (rule fin_image_fin)
  with A1 A2 show thesis
    using FinD func_imagedef by simp
qed
```

If the set defined by a meta-function is finite, then every set defined by a composition of this meta function with another one is finite.

```
lemma Finite1_L6B:
  assumes A1: ∀x∈X. a(x) ∈ Y and A2: {b(y).y∈Y} ∈ Fin(Z)
```

```
  shows {b(a(x)).x∈X} ∈ Fin(Z)
proof -
  from A1 have {b(a(x)).x∈X} ⊆ {b(y).y∈Y} by auto
  with A2 show thesis using Fin_subset_lemma by blast
qed
```

If the set defined by a meta-function is finite, then every set defined by a composition of this meta function with another one is finite.

```
lemma Finite1_L6C:
  assumes A1: ∀y∈Y. b(y) ∈ Z and A2: {a(x). x∈X} ∈ Fin(Y)
  shows {b(a(x)).x∈X} ∈ Fin(Z)
proof -
  let N = {a(x). x∈X}
  from A1 A2 have {b(y). y ∈ N} ∈ Fin(Z)
    by (rule fin_image_fin)
  moreover have {b(a(x)). x∈X} = {b(y). y∈ N}
    by auto
  ultimately show thesis by simp
qed
```

Cartesian product of finite sets is finite.

```
lemma Finite1_L12: assumes A1: A ∈ Fin(A) and A2: B ∈ Fin(B)
  shows A×B ∈ Fin(A×B)
proof -
  have T1:∀a∈A. ∀b∈B. {⟨ a,b⟩} ∈ Fin(A×B) by simp
  have ∀a∈A. {{⟨ a,b⟩}. b ∈ B} ∈ Fin(Fin(A×B))
  proof
    fix a assume A3: a ∈ A
    with T1 have ∀b∈B. {⟨ a,b⟩} ∈ Fin(A×B)
      by simp
    moreover note A2
    ultimately show {{⟨ a,b⟩}. b ∈ B} ∈ Fin(Fin(A×B))
      by (rule fin_image_fin)
  qed
  then have ∀a∈A. ⋃ {{⟨ a,b⟩}. b ∈ B} ∈ Fin(A×B)
    using Fin_UnionI by simp
  moreover have
    ∀a∈A. ⋃ {{⟨ a,b⟩}. b ∈ B} = {a}× B by blast
  ultimately have ∀a∈A. {a}× B ∈ Fin(A×B) by simp
  moreover note A1
  ultimately have {{a}× B. a∈A} ∈ Fin(Fin(A×B))
    by (rule fin_image_fin)
  then have ⋃{{a}× B. a∈A} ∈ Fin(A×B)
    using Fin_UnionI by simp
  moreover have ⋃{{a}× B. a∈A} = A×B by blast
  ultimately show thesis by simp
qed
```

We define the characterisic meta-function that is the identity on a set and assigns a default value everywhere else.

```
definition
  Characteristic(A,default,x) ≡ (if x∈A then x else default)
```

A finite subset is a finite subset of itself.

```
lemma Finite1_L13:
  assumes A1:A ∈ Fin(X) shows A ∈ Fin(A)
proof -
  { assume A=0 hence A ∈ Fin(A) by simp }
  moreover
  { assume A2: A≠0 then obtain c where D1:c∈A
      by auto
    then have ∀x∈X. Characteristic(A,c,x) ∈ A
      using Characteristic_def by simp
    moreover note A1
    ultimately have
      {Characteristic(A,c,x). x∈A} ∈ Fin(A) by (rule fin_image_fin)
    moreover from D1 have
      {Characteristic(A,c,x). x∈A} = A using Characteristic_def by simp
    ultimately have A ∈ Fin(A) by simp }
  ultimately show thesis by blast
qed
```

Cartesian product of finite subsets is a finite subset of cartesian product.

```
lemma Finite1_L14: assumes A1: A ∈ Fin(X) B ∈ Fin(Y)
  shows A×B ∈ Fin(X×Y)
proof -
  from A1 have A×B ⊆ X×Y using FinD by auto
  then have Fin(A×B) ⊆ Fin(X×Y) using Fin_mono by simp
  moreover from A1 have A×B ∈ Fin(A×B)
    using Finite1_L13 Finite1_L12 by simp
  ultimately show thesis by auto
qed
```

The next lemma is needed in the `Group_ZF_3` theory in a couple of places.

```
lemma Finite1_L15:
  assumes A1: {b(x). x∈A} ∈ Fin(B)  {c(x). x∈A} ∈ Fin(C)
  and A2: f : B×C→E
  shows {f⟨ b(x),c(x)⟩. x∈A} ∈ Fin(E)
proof -
  from A1 have {b(x). x∈A}×{c(x). x∈A} ∈ Fin(B×C)
    using Finite1_L14 by simp
  moreover have
    {⟨ b(x),c(x)⟩. x∈A} ⊆ {b(x). x∈A}×{c(x). x∈A}
    by blast
  ultimately have T0: {⟨ b(x),c(x)⟩. x∈A} ∈ Fin(B×C)
    by (rule Fin_subset_lemma)
  with A2 have T1: f{⟨ b(x),c(x)⟩. x∈A} ∈ Fin(E)
```

```
    using Finite1_L6A by auto
  from T0 have ∀x∈A. ⟨ b(x),c(x)⟩ ∈ B×C
    using FinD by auto
  with A2 have
    f{⟨ b(x),c(x)⟩. x∈A} = {f⟨ b(x),c(x)⟩. x∈A}
    using func1_1_L17 by simp
  with T1 show thesis by simp
qed
```

Singletons are in the finite powerset.

```
lemma Finite1_L16: assumes x∈X shows {x} ∈ Fin(X)
  using assms emptyI consI by simp
```

A special case of `Finite1_L15` where the second set is a singleton. In `Group_ZF_3` theory this corresponds to the situation where we multiply by a constant.

```
lemma Finite1_L16AA: assumes {b(x). x∈A} ∈ Fin(B)
  and c∈C and f : B×C→E
  shows {f⟨ b(x),c⟩. x∈A} ∈ Fin(E)
proof -
  from assms have
    ∀y∈B. f⟨y,c⟩ ∈ E
    {b(x). x∈A} ∈ Fin(B)
    using apply_funtype by auto
  then show thesis by (rule Finite1_L6C)
qed
```

First order version of the induction for the finite powerset.

```
lemma Finite1_L16B: assumes A1: P(0) and A2: B∈Fin(X)
  and A3: ∀A∈Fin(X).∀x∈X. x∉A ∧ P(A)⟶P(A∪{x})
  shows P(B)
proof -
  note ‹B∈Fin(X)› and ‹P(0)›
  moreover
  { fix A x
    assume  x ∈ X  A ∈ Fin(X)  x ∉ A  P(A)
    moreover have cons(x,A) = A∪{x} by auto
    moreover note A3
    ultimately have P(cons(x,A)) by simp }
  ultimately show  P(B) by (rule Fin_induct)
qed
```

14.2 Finite range functions

In this section we define functions $f : X \to Y$, with the property that $f(X)$ is a finite subset of Y. Such functions play a important role in the construction of real numbers in the `Real_ZF` series.

Definition of finite range functions.

definition
```
  FinRangeFunctions(X,Y) ≡ {f:X→Y. f(X) ∈ Fin(Y)}
```

Constant functions have finite range.

```
lemma Finite1_L17: assumes c∈Y and X≠0
  shows ConstantFunction(X,c) ∈ FinRangeFunctions(X,Y)
  using assms  func1_3_L1 func_imagedef func1_3_L2 Finite1_L16
    FinRangeFunctions_def by simp
```

Finite range functions have finite range.

```
lemma Finite1_L18: assumes f ∈ FinRangeFunctions(X,Y)
  shows {f(x). x∈X} ∈ Fin(Y)
  using assms FinRangeFunctions_def func_imagedef by simp
```

An alternative form of the definition of finite range functions.

```
lemma Finite1_L19: assumes f:X→Y
  and {f(x). x∈X} ∈ Fin(Y)
  shows f ∈ FinRangeFunctions(X,Y)
  using assms func_imagedef FinRangeFunctions_def by simp
```

A composition of a finite range function with another function is a finite range function.

```
lemma Finite1_L20: assumes A1:f ∈ FinRangeFunctions(X,Y)
  and A2: g : Y→Z
  shows g O f ∈ FinRangeFunctions(X,Z)
proof -
  from A1 A2 have g{f(x). x∈X} ∈ Fin(Z)
    using Finite1_L18 Finite1_L6A
    by simp
  with A1 A2 have {(g O f)(x). x∈X} ∈ Fin(Z)
    using FinRangeFunctions_def apply_funtype
      func1_1_L17 comp_fun_apply by auto
  with A1 A2 show thesis using
    FinRangeFunctions_def comp_fun Finite1_L19
    by auto
qed
```

Image of any subset of the domain of a finite range function is finite.

```
lemma Finite1_L21:
  assumes f ∈ FinRangeFunctions(X,Y) and A⊆X
  shows f(A) ∈ Fin(Y)
proof -
  from assms have f(X) ∈ Fin(Y)  f(A) ⊆ f(X)
    using FinRangeFunctions_def func1_1_L8
    by auto
  then show f(A) ∈ Fin(Y) using Fin_subset_lemma
    by blast
qed
```

end

15 Finite sets 1

theory Finite_ZF_1 **imports** Finite1 Order_ZF_1a

begin

This theory is based on Finite1 theory and is obsolete. It contains properties of finite sets related to order relations. See the FinOrd theory for a better approach.

15.1 Finite vs. bounded sets

The goal of this section is to show that finite sets are bounded and have maxima and minima.

Finite set has a maximum - induction step.

```
lemma Finite_ZF_1_1_L1:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: A∈Fin(X) and A4: x∈X and A5: A=0 ∨ HasAmaximum(r,A)
  shows A∪{x} = 0 ∨ HasAmaximum(r,A∪{x})
proof -
  { assume A=0 then have T1: A∪{x} = {x} by simp
    from A1 have refl(X,r) using total_is_refl by simp
    with T1 A4 have A∪{x} = 0 ∨ HasAmaximum(r,A∪{x})
      using Order_ZF_4_L8 by simp }
  moreover
  { assume A≠0
    with A1 A2 A3 A4 A5 have A∪{x} = 0 ∨ HasAmaximum(r,A∪{x})
      using FinD Order_ZF_4_L9 by simp }
  ultimately show thesis by blast
qed
```

For total and transitive relations finite set has a maximum.

```
theorem Finite_ZF_1_1_T1A:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: B∈Fin(X)
  shows B=0 ∨ HasAmaximum(r,B)
proof -
  have 0=0 ∨ HasAmaximum(r,0) by simp
  moreover note A3
  moreover from A1 A2 have ∀A∈Fin(X). ∀x∈X.
    x∉A ∧ (A=0 ∨ HasAmaximum(r,A)) ⟶ (A∪{x}=0 ∨ HasAmaximum(r,A∪{x}))
    using Finite_ZF_1_1_L1 by simp
  ultimately show  B=0 ∨ HasAmaximum(r,B) by (rule Finite1_L16B)
```

qed

Finite set has a minimum - induction step.

```
lemma Finite_ZF_1_1_L2:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: A∈Fin(X) and A4: x∈X and A5: A=0 ∨ HasAminimum(r,A)
  shows A∪{x} = 0 ∨ HasAminimum(r,A∪{x})
proof -
  { assume A=0 then have T1: A∪{x} = {x} by simp
    from A1 have refl(X,r) using total_is_refl by simp
    with T1 A4 have A∪{x} = 0 ∨ HasAminimum(r,A∪{x})
      using Order_ZF_4_L8 by simp }
  moreover
  { assume A≠0
    with A1 A2 A3 A4 A5 have A∪{x} = 0 ∨ HasAminimum(r,A∪{x})
      using FinD Order_ZF_4_L10 by simp }
  ultimately show thesis by blast
qed
```

For total and transitive relations finite set has a minimum.

```
theorem Finite_ZF_1_1_T1B:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: B ∈ Fin(X)
  shows B=0 ∨ HasAminimum(r,B)
proof -
  have 0=0 ∨ HasAminimum(r,0) by simp
  moreover note A3
  moreover from A1 A2 have ∀A∈Fin(X). ∀x∈X.
    x∉A ∧ (A=0 ∨ HasAminimum(r,A)) ⟶ (A∪{x}=0 ∨ HasAminimum(r,A∪{x}))
    using Finite_ZF_1_1_L2 by simp
  ultimately show  B=0 ∨ HasAminimum(r,B) by (rule Finite1_L16B)
qed
```

For transitive and total relations finite sets are bounded.

```
theorem Finite_ZF_1_T1:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: B∈Fin(X)
  shows IsBounded(B,r)
proof -
  from A1 A2 A3 have B=0 ∨ HasAminimum(r,B) B=0 ∨ HasAmaximum(r,B)
    using Finite_ZF_1_1_T1A Finite_ZF_1_1_T1B by auto
  then have
    B = 0 ∨ IsBoundedBelow(B,r) B = 0 ∨ IsBoundedAbove(B,r)
    using Order_ZF_4_L7 Order_ZF_4_L8A by auto
  then show IsBounded(B,r) using
    IsBounded_def IsBoundedBelow_def IsBoundedAbove_def
    by simp
qed
```

For linearly ordered finite sets maximum and minimum have desired properties. The reason we need linear order is that we need the order to be total and transitive for the finite sets to have a maximum and minimum and then we also need antisymmetry for the maximum and minimum to be unique.

```
theorem Finite_ZF_1_T2:
  assumes A1: IsLinOrder(X,r) and A2: A ∈ Fin(X) and A3: A≠0
  shows
  Maximum(r,A) ∈ A
  Minimum(r,A) ∈ A
  ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
  ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
proof -
  from A1 have T1: r {is total on} X trans(r) antisym(r)
    using IsLinOrder_def by auto
  moreover from T1 A2 A3 have HasAmaximum(r,A)
    using Finite_ZF_1_1_T1A by auto
  moreover from T1 A2 A3 have HasAminimum(r,A)
    using Finite_ZF_1_1_T1B by auto
  ultimately show
    Maximum(r,A) ∈ A
    Minimum(r,A) ∈ A
    ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
    using Order_ZF_4_L3 Order_ZF_4_L4 by auto
qed
```

A special case of `Finite_ZF_1_T2` when the set has three elements.

```
corollary Finite_ZF_1_L2A:
  assumes A1: IsLinOrder(X,r) and A2: a∈X  b∈X  c∈X
  shows
  Maximum(r,{a,b,c}) ∈ {a,b,c}
  Minimum(r,{a,b,c}) ∈ {a,b,c}
  Maximum(r,{a,b,c}) ∈ X
  Minimum(r,{a,b,c}) ∈ X
  ⟨a,Maximum(r,{a,b,c})⟩ ∈ r
  ⟨b,Maximum(r,{a,b,c})⟩ ∈ r
  ⟨c,Maximum(r,{a,b,c})⟩ ∈ r
proof -
  from A2 have I: {a,b,c} ∈ Fin(X)  {a,b,c} ≠ 0
    by auto
  with A1 show II: Maximum(r,{a,b,c}) ∈ {a,b,c}
    by (rule Finite_ZF_1_T2)
  moreover from A1 I show III: Minimum(r,{a,b,c}) ∈ {a,b,c}
    by (rule Finite_ZF_1_T2)
  moreover from A2 have {a,b,c} ⊆ X
    by auto
  ultimately show
    Maximum(r,{a,b,c}) ∈ X
    Minimum(r,{a,b,c}) ∈ X
    by auto
```

```
  from A1 I have ∀x∈{a,b,c}. ⟨x,Maximum(r,{a,b,c})⟩ ∈ r
    by (rule Finite_ZF_1_T2)
  then show
    ⟨a,Maximum(r,{a,b,c})⟩ ∈ r
    ⟨b,Maximum(r,{a,b,c})⟩ ∈ r
    ⟨c,Maximum(r,{a,b,c})⟩ ∈ r
    by auto
qed
```

If for every element of X we can find one in A that is greater, then the A can not be finite. Works for relations that are total, transitive and antisymmetric.

```
lemma Finite_ZF_1_1_L3:
  assumes A1: r {is total on} X
  and A2: trans(r) and A3: antisym(r)
  and A4: r ⊆ X×X and A5: X≠0
  and A6: ∀x∈X. ∃a∈A. x≠a ∧ ⟨x,a⟩ ∈ r
  shows A ∉ Fin(X)
proof -
  from assms have ¬IsBounded(A,r)
    using Order_ZF_3_L14 IsBounded_def
    by simp
  with A1 A2 show A ∉ Fin(X)
    using Finite_ZF_1_T1 by auto
qed

end
```

16 Finite sets and order relations

```
theory FinOrd_ZF imports Finite_ZF func_ZF_1 NatOrder_ZF

begin
```

This theory file contains properties of finite sets related to order relations. Part of this is similar to what is done in `Finite_ZF_1` except that the development is based on the notion of finite powerset defined in `Finite_ZF` rather the one defined in standard Isabelle `Finite` theory.

16.1 Finite vs. bounded sets

The goal of this section is to show that finite sets are bounded and have maxima and minima.

For total and transitive relations nonempty finite set has a maximum.

```
theorem fin_has_max:
  assumes A1: r {is total on} X and A2: trans(r)
```

```
  and A3: B ∈ FinPow(X) and A4: B ≠ 0
  shows HasAmaximum(r,B)
proof -
  have 0=0 ∨ HasAmaximum(r,0) by simp
  moreover have
    ∀A ∈ FinPow(X). A=0 ∨ HasAmaximum(r,A) ⟶
    (∀x∈X. (A ∪ {x}) = 0 ∨ HasAmaximum(r,A ∪ {x}))
  proof -
    { fix A
      assume A ∈ FinPow(X)  A = 0 ∨ HasAmaximum(r,A)
      have ∀x∈X. (A ∪ {x}) = 0 ∨ HasAmaximum(r,A ∪ {x})
      proof -
 { fix x assume x∈X
   note ‹A = 0 ∨ HasAmaximum(r,A)›
   moreover
   { assume A = 0
     then have A∪{x} = {x} by simp
     from A1 have refl(X,r) using total_is_refl
       by simp
     with ‹x∈X› ‹A∪{x} = {x}› have HasAmaximum(r,A∪{x})
       using Order_ZF_4_L8 by simp }
   moreover
   { assume HasAmaximum(r,A)
     with A1 A2 ‹A ∈ FinPow(X)›  ‹x∈X›
     have HasAmaximum(r,A∪{x})
       using FinPow_def Order_ZF_4_L9 by simp }
   ultimately  have A ∪ {x} = 0 ∨ HasAmaximum(r,A ∪ {x})
     by auto
 } thus ∀x∈X. (A ∪ {x}) = 0 ∨ HasAmaximum(r,A ∪ {x})
   by simp
      qed
    } thus thesis by simp
  qed
  moreover note A3
  ultimately have B = 0 ∨  HasAmaximum(r,B)
    by (rule FinPow_induct)
  with A4 show HasAmaximum(r,B) by simp
qed
```

For linearly ordered nonempty finite sets the maximum is in the set and indeed it is the greatest element of the set.

```
lemma linord_max_props: assumes A1: IsLinOrder(X,r) and
  A2: A ∈ FinPow(X) A ≠ 0
  shows
  Maximum(r,A) ∈ A
  Maximum(r,A) ∈ X
  ∀a∈A. ⟨a,Maximum(r,A)⟩ ∈ r
proof -
  from A1 A2 show
```

```
    Maximum(r,A) ∈ A and ∀a∈A. ⟨a,Maximum(r,A)⟩ ∈ r
    using IsLinOrder_def fin_has_max Order_ZF_4_L3
    by auto
  with A2 show Maximum(r,A) ∈ X using FinPow_def
    by auto
qed
```

Every nonempty subset of a natural number has a maximum with expected properties.

```
lemma nat_max_props: assumes n∈nat A⊆n A≠0
  shows
  Maximum(Le,A) ∈ A
  Maximum(Le,A) ∈ nat
  ∀k∈A. k ≤ Maximum(Le,A)
proof -
  from assms(1,2) have A ∈ FinPow(nat)
    using nat_finpow_nat subset_finpow by blast
  with assms(3) show
    Maximum(Le,A) ∈ A
    Maximum(Le,A) ∈ nat
    using NatOrder_ZF_1_L2(4) linord_max_props(1,2) by simp_all
  from assms(3) ‹A ∈ FinPow(nat)› have ∀k∈A. ⟨k,Maximum(Le,A)⟩ ∈ Le
    using linord_max_props NatOrder_ZF_1_L2(4) by blast
  then show  ∀k∈A. k ≤ Maximum(Le,A) by simp
qed
```

Yet another version of induction where the induction step is valid only up to $n \in \mathbb{N}$ rather than for all natural numbers. This lemma is redundant as it is easier to prove this assertion using lemma `fin_nat_ind` from `Nat_ZF_IML` which was done in lemma `fin_nat_ind1` there. It is left here for now as an alternative proof based on properties of the maximum of a finite set.

```
lemma ind_on_nat2:
  assumes n∈nat and P(0) and ∀j∈n. P(j)⟶P(j #+ 1)
  shows ∀j∈n #+ 1. P(j) and P(n)
proof -
  let A = {k∈succ(n). ∀j∈succ(k). P(j)}
  let M = Maximum(Le,A)
  from assms(1,2) have I: succ(n) ∈ nat A⊆succ(n) A≠0
    using empty_in_every_succ by auto
  then have M ∈ A by (rule nat_max_props)
  have n=M
  proof -
    from ‹M ∈ A› have M ∈ succ(n) by blast
    with assms(1) have M∈n ∨ M=n by auto
    moreover
    { assume M ∈ n
      from I have M ∈ nat by (rule nat_max_props)
      from assms(3) ‹M∈A› ‹M∈n› have P(M #+ 1) by blast
```

```
      with ‹M ∈ nat› have P(succ(M)) using succ_add_one(1) by simp
      with ‹M∈A› have ∀j∈succ(succ(M)). P(j) by blast
      moreover from assms(1) ‹M ∈ n› have succ(M) ∈ succ(n)
        using succ_ineq1 by simp
      moreover from I have ∀k∈A. k ≤ M
        by (rule nat_max_props)
      ultimately have False by blast
    }
    ultimately show n=M by auto
  qed
  with ‹M ∈ A› have n∈A by (rule eq_mem)
  with assms(1) show ∀j∈n #+ 1. P(j) and P(n)
    using succ_add_one(1) by simp_all
qed
```

16.2 Order isomorphisms of finite sets

In this section we establish that if two linearly ordered finite sets have the same number of elements, then they are order-isomorphic and the isomorphism is unique. This allows us to talk about "enumeration" of a linearly ordered finite set. We define the enumeration as the order isomorphism between the number of elements of the set (which is a natural number $n = \{0, 1, .., n-1\}$) and the set.

A really weird corner case - empty set is order isomorphic with itself.

```
lemma empty_ord_iso: shows ord_iso(0,r,0,R) ≠ 0
proof -
  have 0 ≈ 0 using eqpoll_refl by simp
  then obtain f where f ∈ bij(0,0)
    using eqpoll_def by blast
  then show thesis using ord_iso_def by auto
qed
```

Even weirder than `empty_ord_iso` The order automorphism of the empty set is unique.

```
lemma empty_ord_iso_uniq:
  assumes f ∈ ord_iso(0,r,0,R)  g ∈ ord_iso(0,r,0,R)
  shows f = g
proof -
  from assms have f : 0 → 0 and g: 0 → 0
    using ord_iso_def bij_def surj_def by auto
    moreover have ∀x∈0. f(x) = g(x) by simp
    ultimately show f = g by (rule func_eq)
qed
```

The empty set is the only order automorphism of itself.

```
lemma empty_ord_iso_empty: shows ord_iso(0,r,0,R) = {0}
proof -
```

```
  have 0 ∈ ord_iso(0,r,0,R)
  proof -
    have ord_iso(0,r,0,R) ≠ 0 by (rule empty_ord_iso)
    then obtain f where f ∈ ord_iso(0,r,0,R) by auto
    then show 0 ∈ ord_iso(0,r,0,R)
      using ord_iso_def bij_def surj_def fun_subset_prod
      by auto
  qed
  then show ord_iso(0,r,0,R) = {0} using empty_ord_iso_uniq
    by blast
qed
```

An induction (or maybe recursion?) scheme for linearly ordered sets. The induction step is that we show that if the property holds when the set is a singleton or for a set with the maximum removed, then it holds for the set. The idea is that since we can build any finite set by adding elements on the right, then if the property holds for the empty set and is invariant with respect to this operation, then it must hold for all finite sets.

```
lemma fin_ord_induction:
  assumes A1: IsLinOrder(X,r) and A2: P(0) and
  A3: ∀A ∈ FinPow(X). A ≠ 0 ⟶ (P(A - {Maximum(r,A)}) ⟶ P(A))
  and A4: B ∈ FinPow(X) shows P(B)
proof -
  note A2
  moreover have ∀ A ∈ FinPow(X). A ≠ 0 ⟶ (∃a∈A. P(A-{a}) ⟶ P(A))
  proof -
    { fix A assume A ∈  FinPow(X) and A ≠ 0
      with A1 A3 have ∃a∈A. P(A-{a}) ⟶ P(A)
 using IsLinOrder_def fin_has_max
   IsLinOrder_def Order_ZF_4_L3
 by blast
    } thus thesis by simp
  qed
  moreover note A4
  ultimately show P(B) by (rule FinPow_ind_rem_one)
qed
```

A sligltly more complicated version of `fin_ord_induction` that allows to prove properties that are not true for the empty set.

```
lemma fin_ord_ind:
  assumes A1: IsLinOrder(X,r) and A2: ∀A ∈ FinPow(X).
  A = 0 ∨ (A = {Maximum(r,A)} ∨ P(A - {Maximum(r,A)}) ⟶ P(A))
  and A3: B ∈  FinPow(X) and A4: B≠0
  shows P(B)
proof -
  { fix A assume A ∈  FinPow(X) and A ≠ 0
    with A1 A2 have
      ∃a∈A. A = {a} ∨ P(A-{a}) ⟶ P(A)
```

```
      using IsLinOrder_def fin_has_max
 IsLinOrder_def Order_ZF_4_L3
      by blast
  } then have ∀A ∈ FinPow(X).
      A = 0 ∨ (∃a∈A. A = {a} ∨ P(A-{a}) ⟶ P(A))
    by auto
  with A3 A4 show P(B) using  FinPow_rem_ind
    by simp
qed
```

Yet another induction scheme. We build a linearly ordered set by adding elements that are greater than all elements in the set.

```
lemma fin_ind_add_max:
  assumes A1: IsLinOrder(X,r) and A2: P(0) and A3: ∀ A ∈ FinPow(X).

  ( ∀ x ∈ X-A. P(A) ∧ (∀a∈A. ⟨a,x⟩ ∈ r ) ⟶ P(A ∪ {x}))
  and A4: B ∈ FinPow(X)
  shows P(B)
proof -
  note A1 A2
  moreover have
    ∀C ∈ FinPow(X). C ≠ 0 ⟶ (P(C - {Maximum(r,C)}) ⟶ P(C))
    proof -
      { fix C assume C ∈ FinPow(X) and C ≠ 0
 let x = Maximum(r,C)
 let A = C - {x}
 assume P(A)
 moreover from ‹C ∈ FinPow(X)› have A ∈ FinPow(X)
   using fin_rem_point_fin by simp
 moreover from A1 ‹C ∈ FinPow(X)› ‹C ≠ 0› have
   x ∈ C and x ∈ X - A and ∀a∈A. ⟨a,x⟩ ∈ r
   using linord_max_props by auto
 moreover note A3
 ultimately have P(A ∪ {x}) by auto
 moreover from ‹x ∈ C› have A ∪ {x} = C
   by auto
 ultimately have P(C) by simp
      } thus thesis by simp
    qed
    moreover note A4
  ultimately show P(B) by (rule fin_ord_induction)
qed
```

The only order automorphism of a linearly ordered finite set is the identity.

```
theorem fin_ord_auto_id: assumes A1: IsLinOrder(X,r)
  and A2: B ∈  FinPow(X) and A3: B≠0
  shows ord_iso(B,r,B,r) = {id(B)}
proof -
  note A1
```

```
 moreover
 { fix A assume A ∈  FinPow(X) A≠0
   let M = Maximum(r,A)
   let A₀ = A - {M}
   assume A = {M} ∨ ord_iso(A₀,r,A₀,r) = {id(A₀)}
   moreover
   { assume A = {M}
     have ord_iso({M},r,{M},r) = {id({M})}
using id_ord_auto_singleton by simp
     with ‹A = {M}› have ord_iso(A,r,A,r) = {id(A)}
by simp }
   moreover
   { assume ord_iso(A₀,r,A₀,r) = {id(A₀)}
     have ord_iso(A,r,A,r) = {id(A)}
     proof
show {id(A)} ⊆ ord_iso(A,r,A,r)
  using id_ord_iso by simp
{ fix f assume f ∈ ord_iso(A,r,A,r)
  with A1 ‹A ∈  FinPow(X)› ‹A≠0› have
    restrict(f,A₀) ∈ ord_iso(A₀, r, A-{f(M)},r)
    using IsLinOrder_def fin_has_max ord_iso_rem_max
    by auto
  with A1 ‹A ∈  FinPow(X)› ‹A≠0› ‹f ∈ ord_iso(A,r,A,r)›
    ‹ord_iso(A₀,r,A₀,r) = {id(A₀)}›
  have restrict(f,A₀) = id(A₀)
    using IsLinOrder_def fin_has_max max_auto_fixpoint
    by auto
  moreover from A1 ‹f ∈ ord_iso(A,r,A,r)›
    ‹A ∈  FinPow(X)› ‹A≠0› have
    f : A → A and M ∈ A and f(M) = M
    using ord_iso_def bij_is_fun IsLinOrder_def
      fin_has_max Order_ZF_4_L3 max_auto_fixpoint
    by auto
  ultimately have f = id(A) using id_fixpoint_rem
    by simp
} then show ord_iso(A,r,A,r) ⊆ {id(A)}
  by auto
     qed
   }
   ultimately have ord_iso(A,r,A,r) = {id(A)}
     by auto
 } then have ∀A ∈  FinPow(X). A = 0 ∨
     (A = {Maximum(r,A)} ∨
     ord_iso(A-{Maximum(r,A)},r,A-{Maximum(r,A)},r) =
     {id(A-{Maximum(r,A)})} ⟶  ord_iso(A,r,A,r) = {id(A)})
   by auto
 moreover note A2 A3
 ultimately show ord_iso(B,r,B,r) = {id(B)}
   by (rule fin_ord_ind)
```

qed

Every two finite linearly ordered sets are order isomorphic. The statement is formulated to make the proof by induction on the size of the set easier, see `fin_ord_iso_ex` for an alternative formulation.

```
lemma fin_order_iso:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: n ∈ nat
  shows ∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
  A ≈ n ∧ B ≈ n ⟶ ord_iso(A,r,B,R) ≠ 0
proof -
  note A2
  moreover have ∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
    A ≈ 0 ∧ B ≈ 0 ⟶ ord_iso(A,r,B,R) ≠ 0
    using eqpoll_0_is_0 empty_ord_iso by blast
  moreover have ∀k ∈ nat.
    (∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
    A ≈ k ∧ B ≈ k ⟶ ord_iso(A,r,B,R) ≠ 0) ⟶
    (∀C ∈ FinPow(X). ∀D ∈ FinPow(Y).
    C ≈ succ(k) ∧ D ≈ succ(k) ⟶ ord_iso(C,r,D,R) ≠ 0)
  proof -
    { fix k assume k ∈ nat
      assume A3: ∀A ∈ FinPow(X). ∀B ∈ FinPow(Y).
 A ≈ k ∧ B ≈ k ⟶ ord_iso(A,r,B,R) ≠ 0
      have ∀C ∈ FinPow(X). ∀D ∈ FinPow(Y).
 C ≈ succ(k) ∧ D ≈ succ(k) ⟶ ord_iso(C,r,D,R) ≠ 0
      proof -
 { fix C assume C ∈ FinPow(X)
   fix D assume D ∈ FinPow(Y)
   assume C ≈ succ(k)  D ≈ succ(k)
   then have C ≠ 0 and D≠ 0
     using eqpoll_succ_imp_not_empty by auto
   let M_C = Maximum(r,C)
   let M_D = Maximum(R,D)
   let C_0 = C - {M_C}
   let D_0 = D - {M_D}
   from ‹C ∈ FinPow(X)› have C ⊆ X
     using FinPow_def by simp
   with A1 have IsLinOrder(C,r)
     using ord_linear_subset by blast
   from ‹D ∈ FinPow(Y)› have D ⊆ Y
     using FinPow_def by simp
   with A1 have IsLinOrder(D,R)
     using ord_linear_subset by blast
   from A1 ‹C ∈ FinPow(X)› ‹D ∈ FinPow(Y)›
     ‹C ≠ 0› ‹D≠ 0› have
     HasAmaximum(r,C) and HasAmaximum(R,D)
     using IsLinOrder_def fin_has_max
     by auto
```

```
    with A1 have M_C ∈ C and M_D ∈ D
      using IsLinOrder_def Order_ZF_4_L3 by auto
    with ‹C ≈ succ(k)›  ‹D ≈ succ(k)› have
      C_0  ≈ k and D_0 ≈ k using Diff_sing_eqpoll by auto
    from ‹C ∈ FinPow(X)› ‹D ∈ FinPow(Y)›
    have C_0 ∈  FinPow(X) and D_0 ∈  FinPow(Y)
      using fin_rem_point_fin by auto
    with A3 ‹C_0  ≈ k› ‹D_0 ≈ k› have
      ord_iso(C_0,r,D_0,R) ≠ 0 by simp
    with ‹IsLinOrder(C,r)›  ‹IsLinOrder(D,R)›
      ‹HasAmaximum(r,C)› ‹HasAmaximum(R,D)›
    have ord_iso(C,r,D,R) ≠ 0
      by (rule rem_max_ord_iso)
  } thus thesis by simp
      qed
    } thus thesis by blast
  qed
  ultimately show thesis by (rule ind_on_nat)
qed
```

Every two finite linearly ordered sets are order isomorphic.

```
lemma fin_ord_iso_ex:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: A ∈ FinPow(X) B ∈ FinPow(Y) and A3: B ≈ A
  shows ord_iso(A,r,B,R) ≠ 0
proof -
  from A2 obtain n where n ∈ nat and A ≈ n
    using finpow_decomp by auto
  from  A3 ‹A ≈ n› have B ≈ n by (rule eqpoll_trans)
  with A1 A2 ‹A ≈ n› ‹n ∈ nat› show ord_iso(A,r,B,R) ≠ 0
    using fin_order_iso by simp
qed
```

Existence and uniqueness of order isomorphism for two linearly ordered sets with the same number of elements.

```
theorem fin_ord_iso_ex_uniq:
  assumes A1: IsLinOrder(X,r)  IsLinOrder(Y,R) and
  A2: A ∈ FinPow(X) B ∈ FinPow(Y) and A3: B ≈ A
  shows ∃!f. f ∈ ord_iso(A,r,B,R)
proof
  from assms show ∃f. f ∈ ord_iso(A,r,B,R)
    using fin_ord_iso_ex by blast
  fix f g
  assume A4: f ∈ ord_iso(A,r,B,R)  g ∈ ord_iso(A,r,B,R)
  then have converse(g) ∈ ord_iso(B,R,A,r)
    using ord_iso_sym by simp
  with ‹f ∈ ord_iso(A,r,B,R)› have
    I: converse(g) O f ∈  ord_iso(A,r,A,r)
    by (rule ord_iso_trans)
```

```
  { assume A ≠ 0
    with A1 A2 I have converse(g) O f = id(A)
      using fin_ord_auto_id by auto
    with A4 have f = g
      using ord_iso_def comp_inv_id_eq_bij by auto }
  moreover
  { assume A = 0
    then have A ≈ 0 using eqpoll_0_iff
      by simp
    with A3 have B ≈ 0 by (rule eqpoll_trans)
    with A4 ‹A = 0› have
      f ∈ ord_iso(0,r,0,R) and  g ∈ ord_iso(0,r,0,R)
      using eqpoll_0_iff by auto
    then have f = g by (rule empty_ord_iso_uniq) }
  ultimately show f = g
    using ord_iso_def comp_inv_id_eq_bij
    by auto
qed

end
```

17 Cardinal numbers

```
theory Cardinal_ZF imports ZF.CardinalArith Finite_ZF func1

begin
```

This theory file deals with results on cardinal numbers (cardinals). Cardinals are a genaralization of the natural numbers, used to measure the cardinality (size) of sets. Contributed by Daniel de la Concepcion.

17.1 Some new ideas on cardinals

All the results of this section are done without assuming the Axiom of Choice. With the Axiom of Choice in play, the proofs become easier and some of the assumptions may be dropped.

Since General Topology Theory is closely related to Set Theory, it is very interesting to make use of all the possibilities of Set Theory to try to classify homeomorphic topological spaces. These ideas are generally used to prove that two topological spaces are not homeomorphic.

There exist cardinals which are the successor of another cardinal, but; as happens with ordinals, there are cardinals which are limit cardinal.

```
definition
    LimitC(i) ≡ Card(i) ∧ 0<i ∧ (∀y. (y<i∧Card(y)) ⟶ csucc(y)<i)
```

Simple fact used a couple of times in proofs.

```
lemma nat_less_infty: assumes n∈nat and InfCard(X) shows n<X
proof -
  from assms have n<nat and nat≤X using lt_def InfCard_def by auto
  then show n<X using lt_trans2 by blast
qed
```

There are three types of cardinals, the zero one, the succesors of other cardinals and the limit cardinals.

```
lemma Card_cases_disj:
  assumes Card(i)
  shows i=0 | (∃j. Card(j) ∧ i=csucc(j)) | LimitC(i)
proof-
  from assms have D: Ord(i) using Card_is_Ord by auto
  {
    assume F: i≠0
    assume Contr: ~LimitC(i)
    from F D have 0<i using Ord_0_lt by auto
    with Contr assms have ∃y. y < i ∧ Card(y) ∧ ¬ csucc(y) < i
      using LimitC_def by blast
    then obtain y where  y < i ∧ Card(y) ∧ ¬ csucc(y) < i  by blast
    with D have  y < i  i≤csucc(y) and O: Card(y)
      using not_lt_imp_le lt_Ord Card_csucc Card_is_Ord
      by auto
    with assms have csucc(y)≤ii≤csucc(y) using csucc_le by auto
    then have i=csucc(y) using le_anti_sym by auto
    with O have ∃j. Card(j) ∧ i=csucc(j) by auto
  } thus thesis by auto
qed
```

Given an ordinal bounded by a cardinal in ordinal order, we can change to the order of sets.

```
lemma le_imp_lesspoll:
  assumes Card(Q)
  shows A ≤ Q ⟹ A ≲ Q
proof -
  assume A ≤ Q
  then have A<Q∨A=Q using le_iff by auto
  then have A≈Q∨A< Q using eqpoll_refl by auto
  with assms have A≈Q∨A≺ Q using lt_Card_imp_lesspoll by auto
  then show A≲Q using lesspoll_def eqpoll_imp_lepoll by auto
qed
```

There are two types of infinite cardinals, the natural numbers and those that have at least one infinite strictly smaller cardinal.

```
lemma InfCard_cases_disj:
  assumes InfCard(Q)
  shows Q=nat ∨ (∃j. csucc(j)≲Q ∧ InfCard(j))
proof-
```

```
  {
    assume ∀j. ¬ csucc(j) ≲ Q ∨ ¬ InfCard(j)
    then have D: ¬ csucc(nat) ≲ Q using InfCard_nat by auto
    with D assms have ¬(csucc(nat) ≤ Q) using le_imp_lesspoll InfCard_is_Card

      by auto
    with assms have Q<(csucc(nat))
      using not_le_iff_lt Card_is_Ord Card_csucc Card_is_Ord
        Card_is_Ord InfCard_is_Card Card_nat by auto
    with assms have Q≤nat using Card_lt_csucc_iff InfCard_is_Card Card_nat

      by auto
    with assms have Q=nat using InfCard_def le_anti_sym by auto
  }
  thus thesis by auto
qed
```

A more readable version of standard Isabelle/ZF `Ord_linear_lt`

```
lemma Ord_linear_lt_IML: assumes Ord(i) Ord(j)
  shows i<j ∨ i=j ∨ j<i
  using assms lt_def Ord_linear disjE by simp
```

A set is injective and not bijective to the successor of a cardinal if and only if it is injective and possibly bijective to the cardinal.

```
lemma Card_less_csucc_eq_le:
  assumes Card(m)
  shows A ≺ csucc(m) ⟷ A ≲ m
proof
  have S: Ord(csucc(m)) using Card_csucc Card_is_Ord assms by auto
  {
    assume A: A ≺ csucc(m)
    with S have |A|≈A using lesspoll_imp_eqpoll by auto
    also from A have ...≺ csucc(m) by auto
    finally have |A|≺ csucc(m) by auto
    then have |A|≲csucc(m)~(|A|≈csucc(m)) using lesspoll_def by auto
    with S have ||A||≤csucc(m)|A|≠csucc(m) using lepoll_cardinal_le by
auto
    then have |A|≤csucc(m) |A|≠csucc(m) using Card_def Card_cardinal
by auto
    then have I: ~(csucc(m)<|A|) |A|≠csucc(m) using le_imp_not_lt by
auto
    from S have csucc(m)<|A| ∨ |A|=csucc(m) ∨ |A|<csucc(m)
      using Card_cardinal Card_is_Ord Ord_linear_lt_IML by auto
    with I have |A|<csucc(m) by simp
    with assms have |A|≤m using Card_lt_csucc_iff Card_cardinal
      by auto
    then have |A|=m∨ |A|< m using le_iff by auto
    then have |A|≈m∨|A|< m using eqpoll_refl by auto
    then have |A|≈m∨|A|≺ m using lt_Card_imp_lesspoll assms by auto
```

```
    then have T:|A|≲m using lesspoll_def eqpoll_imp_lepoll by auto
    from A S have A≈|A| using lesspoll_imp_eqpoll eqpoll_sym by auto
    also from T have ...≲m by auto
    finally show A≲m by simp
  }
  {
    assume A: A≲m
    from assms have m≺csucc(m) using lt_Card_imp_lesspoll Card_csucc
Card_is_Ord
      lt_csucc by auto
    with A show A≺csucc(m) using lesspoll_trans1 by auto
  }
qed
```

If the successor of a cardinal is infinite, so is the original cardinal.

```
lemma csucc_inf_imp_inf:
  assumes Card(j) and InfCard(csucc(j))
  shows InfCard(j)
proof-
  {
    assume f:Finite (j)
    then obtain n where n∈nat j≈n using Finite_def by auto
    with assms(1) have TT: j=n n∈nat
      using cardinal_cong nat_into_Card Card_def by auto
    then have Q:succ(j)∈nat using nat_succI by auto
    with f TT have T: Finite(succ(j)) Card(succ(j))
      using nat_into_Card nat_succI by auto
    from T(2) have Card(succ(j))∧ j<succ(j) using Card_is_Ord by auto
    moreover from this have Ord(succ(j)) using Card_is_Ord by auto
    moreover
    { fix x
      assume A: x<succ(j)
      {
        assume Card(x)∧ j<x
        with A have False using lt_trans1 by auto
      }
      hence ~(Card(x)∧ j<x) by auto
    }
    ultimately have (μ L. Card(L) ∧ j < L)=succ(j)
      by (rule Least_equality)
    then have csucc(j)=succ(j) using csucc_def by auto
    with Q have csucc(j)∈nat by auto
    then have csucc(j)<nat using lt_def Card_nat Card_is_Ord by auto
    with assms(2) have False using InfCard_def lt_trans2 by auto
  }
  then have ~(Finite (j)) by auto
  with assms(1) show thesis using Inf_Card_is_InfCard by auto
qed
```

Since all the cardinals previous to `nat` are finite, it cannot be a successor cardinal; hence it is a LimitC cardinal.

```
corollary LimitC_nat:
  shows LimitC(nat)
proof-
  note Card_nat
  moreover have 0<nat using lt_def by auto
  moreover
  {
    fix y
    assume AS: y<natCard(y)
    then have ord: Ord(y) unfolding lt_def by auto
    then have Cacsucc: Card(csucc(y)) using Card_csucc by auto
    {
      assume nat≤csucc(y)
      with Cacsucc have InfCard(csucc(y)) using InfCard_def by auto
      with AS(2) have InfCard(y) using csucc_inf_imp_inf by auto
      then have nat≤y using InfCard_def by auto
      with AS(1) have False using lt_trans2 by auto
    }
    hence ~(nat≤csucc(y)) by auto
    then have csucc(y)<nat using not_le_iff_lt Ord_nat Cacsucc Card_is_Ord
by auto
  }
  ultimately show thesis using LimitC_def by auto
qed
```

17.2 Main result on cardinals (without the Axiom of Choice)

If two sets are strictly injective to an infinite cardinal, then so is its union. For the case of successor cardinal, this theorem is done in the isabelle library in a more general setting; but that theorem is of not use in the case where LimitC(Q) and it also makes use of the Axiom of Choice. The mentioned theorem is in the theory file Cardinal_AC.thy

Note that if Q is finite and different from 1, let's assume $Q = n$, then the union of A and B is not bounded by Q. Counterexample: two disjoint sets of $n-1$ elements each have a union of $2n-2$ elements which are more than n.

Note also that if $Q = 1$ then A and B must be empty and the union is then empty too; and Q cannot be 0 because no set is injective and not bijective to 0.

The proof is divided in two parts, first the case when both sets A and B are finite; and second, the part when at least one of them is infinite. In the first part, it is used the fact that a finite union of finite sets is finite. In the second part it is used the linear order on cardinals (ordinals). This proof can not be generalized to a setting with an infinite union easily.

```
lemma less_less_imp_un_less:
  assumes A≺Q and B≺Q and InfCard(Q)
  shows A ∪ B≺Q
proof-
{
  assume Finite (A) ∧ Finite(B)
  then have Finite(A ∪ B) using Finite_Un by auto
  then obtain n where R: A ∪ B ≈n  n∈nat using Finite_def
    by auto
  then have |A ∪ B|<nat using lt_def  cardinal_cong
    nat_into_Card  Card_def  Card_nat Card_is_Ord by auto
  with assms(3) have T: |A ∪ B|<Q using InfCard_def lt_trans2 by auto
  from R have Ord(n)A ∪ B ≲ n using nat_into_Card Card_is_Ord eqpoll_imp_lepoll
by auto
  then have A ∪ B≈|A ∪ B| using lepoll_Ord_imp_eqpoll eqpoll_sym by
auto
  also from T assms(3) have ...≺Q using lt_Card_imp_lesspoll InfCard_is_Card
    by auto
  finally have A ∪ B≺Q by simp
}
moreover
{
  assume ~(Finite (A) ∧ Finite(B))
  hence A: ~Finite (A) ∨ ~Finite(B) by auto
  from assms have B: |A|≈A |B|≈B using lesspoll_imp_eqpoll lesspoll_imp_eqpoll
    InfCard_is_Card Card_is_Ord by auto
  from B(1) have Aeq: ∀x. (|A|≈x) ⟶ (A≈x)
    using eqpoll_sym eqpoll_trans by blast
  from B(2) have Beq: ∀x. (|B|≈x) ⟶ (B≈x)
    using eqpoll_sym eqpoll_trans by blast
  with A Aeq have ~Finite(|A|)∨ ~Finite(|B|) using Finite_def
    by auto
  then have D: InfCard(|A|)∨InfCard(|B|)
    using Inf_Card_is_InfCard Inf_Card_is_InfCard  Card_cardinal by blast
  {
    assume AS: |A| < |B|
    {
      assume ~InfCard(|A|)
      with D have InfCard(|B|) by auto
    }
    moreover
    {
      assume InfCard(|A|)
      then have nat≤|A| using InfCard_def by auto
      with AS have nat<|B| using lt_trans1 by auto
      then have nat≤|B| using leI by auto
      then have InfCard(|B|) using InfCard_def Card_cardinal by auto
    }
    ultimately have INFB: InfCard(|B|) by auto
```

```
    then have 2<|B| using nat_less_infty by simp
    then have AG: 2≲|B| using lt_Card_imp_lesspoll Card_cardinal lesspoll_def
      by auto
    from B(2) have |B|≈B by simp
    also from assms(2) have ...≺Q by auto
    finally have TTT: |B|≺Q by simp
    from B(1) have Card(|B|) A ≲|A| using eqpoll_sym Card_cardinal eqpoll_imp_lepoll

      by auto
    with AS have A≺|B| using  lt_Card_imp_lesspoll lesspoll_trans1 by
auto
    then have I1: A≲|B| using lesspoll_def by auto
    from B(2) have I2: B≲|B| using  eqpoll_sym eqpoll_imp_lepoll by
auto
    have A ∪ B≲A+B using Un_lepoll_sum by auto
    also from I1 I2 have ...≲ |B| + |B| using sum_lepoll_mono by auto
    also from AG have ...≲|B| * |B| using sum_lepoll_prod by auto
    also from assms(3) INFB have ...≈|B| using InfCard_square_eqpoll
      by auto
    finally have A ∪ B≲|B| by simp
    also from TTT have ...≺Q by auto
    finally have A ∪ B≺Q by simp
  }
  moreover
  {
    assume AS: |B| < |A|
    {
      assume ~InfCard(|B|)
      with D have InfCard(|A|) by auto
    }
    moreover
    {
      assume InfCard(|B|)
      then have nat≤|B| using InfCard_def by auto
      with AS have nat<|A| using lt_trans1 by auto
      then have nat≤|A| using leI by auto
      then have InfCard(|A|) using InfCard_def Card_cardinal by auto
    }
    ultimately have INFB: InfCard(|A|) by auto
    then have 2<|A| using nat_less_infty by simp
    then have AG: 2≲|A| using lt_Card_imp_lesspoll Card_cardinal lesspoll_def
        by auto
    from B(1) have |A|≈A by simp
    also from assms(1) have ...≺Q by auto
    finally have TTT: |A|≺Q by simp
    from B(2) have Card(|A|) B ≲|B| using eqpoll_sym Card_cardinal eqpoll_imp_lepoll

      by auto
    with AS have B≺|A| using  lt_Card_imp_lesspoll lesspoll_trans1 by
```

```
auto
    then have I1: B≲|A| using lesspoll_def by auto
    from B(1) have I2: A≲|A| using eqpoll_sym eqpoll_imp_lepoll by auto
    have A ∪ B≲A+B using Un_lepoll_sum by auto
    also from I1 I2 have ...≲ |A| + |A| using sum_lepoll_mono by auto
    also from AG have ...≲|A| * |A| using sum_lepoll_prod by auto
    also from INFB assms(3) have ...≈|A| using InfCard_square_eqpoll
       by auto
    finally have A ∪ B≲|A| by simp
    also from TTT have ...≺Q by auto
    finally have A ∪ B≺Q by simp
    }
    moreover
    {
      assume AS: |A|=|B|
      with D have INFB: InfCard(|A|) by auto
      then have 2<|A| using nat_less_infty by simp
      then have AG: 2≲|A| using lt_Card_imp_lesspoll Card_cardinal us-
ing lesspoll_def
        by auto
      from B(1) have |A|≈A by simp
      also from assms(1) have ...≺Q by auto
      finally have TTT: |A|≺Q by simp
      from AS B have I1: A≲|A|and I2:B≲|A| using eqpoll_refl eqpoll_imp_lepoll
        eqpoll_sym by auto
      have A ∪ B≲A+B using Un_lepoll_sum by auto
      also from I1 I2 have ...≲ |A| + |A| using sum_lepoll_mono by auto
      also from AG have ...≲|A| * |A| using sum_lepoll_prod  by auto
      also from assms(3) INFB have ...≈|A| using InfCard_square_eqpoll
        by auto
      finally have A ∪ B≲|A| by simp
      also from TTT have ...≺Q by auto
      finally have A ∪ B≺Q by simp
    }
    ultimately have A ∪ B≺Q using Ord_linear_lt_IML Card_cardinal Card_is_Ord
by auto
  }
  ultimately show A ∪ B≺Q by auto
qed
```

17.3 Choice axioms

We want to prove some theorems assuming that some version of the Axiom of Choice holds. To avoid introducing it as an axiom we will define an appropriate predicate and put that in the assumptions of the theorems. That way technically we stay inside ZF.

The first predicate we define states that the axiom of Q-choice holds for subsets of K if we can find a choice function for every family of subsets of

K whose (that family's) cardinality does not exceed Q.

definition
```
  AxiomCardinalChoice ({the axiom of}_{choice holds for subsets}_) where
  {the axiom of} Q {choice holds for subsets}K ≡ Card(Q) ∧ (∀ M N. (M
≲Q ∧  (∀t∈M. Nt≠0 ∧ Nt⊆K)) ⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)))
```

Next we define a general form of Q choice where we don't require a collection of sets to be included in a set.

definition
```
  AxiomCardinalChoiceGen ({the axiom of}_{choice holds}) where
  {the axiom of} Q {choice holds} ≡ Card(Q) ∧ (∀ M N. (M ≲Q ∧  (∀t∈M.
Nt≠0)) ⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)))
```

The axiom of choice holds if and only if the `AxiomCardinalChoice` holds for every couple of a cardinal `Q` and a set `K`.

```
lemma choice_subset_imp_choice:
  shows {the axiom of} Q {choice holds} ⟷ (∀ K. {the axiom of} Q {choice
holds for subsets}K)
  unfolding AxiomCardinalChoice_def AxiomCardinalChoiceGen_def by blast
```

A choice axiom for greater cardinality implies one for smaller cardinality

```
lemma greater_choice_imp_smaller_choice:
  assumes Q≲Q1 Card(Q)
  shows {the axiom of} Q1 {choice holds} ⟶ ({the axiom of} Q {choice
holds}) using assms
  AxiomCardinalChoiceGen_def lepoll_trans by auto
```

If we have a surjective function from a set which is injective to a set of ordinals, then we can find an injection which goes the other way.

```
lemma surj_fun_inv:
  assumes f ∈ surj(A,B) A⊆Q Ord(Q)
  shows B≲A
proof-
  let g = {⟨m,μ j. j∈A ∧ f(j)=m⟩. m∈B}
  have g:B→range(g) using lam_is_fun_range by simp
  then have fun: g:B→g(B) using range_image_domain by simp
  from assms(2,3) have OA: ∀j∈A. Ord(j) using lt_def Ord_in_Ord by auto
  {
    fix x
    assume x∈g(B)
    then have x∈range(g) and ∃y∈B. ⟨y,x⟩∈g by auto
    then obtain y where T: x=(μ j. j∈A∧ f(j)=y) and y∈B by auto
    with assms(1) OA obtain z where P: z∈A ∧ f(z)=y Ord(z) unfolding
surj_def
      by auto
    with T have x∈A ∧ f(x)=y using LeastI by simp
    hence x∈A by simp
  }
```

```
  then have g(B) ⊆ A by auto
  with fun have fun2: g:B→A using fun_weaken_type by auto
  then have g∈inj(B,A)
  proof -
    {
      fix w x
      assume AS: gw=gx w∈B x∈B
      from assms(1) OA AS(2,3) obtain wz xz where
        P1: wz∈A ∧ f(wz)=w  Ord(wz) and P2: xz∈A ∧ f(xz)=x  Ord(xz)

        unfolding surj_def by blast
      from P1 have (μ j. j∈A∧ fj=w) ∈ A ∧ f(μ j. j∈A∧ fj=w)=w
        by (rule LeastI)
      moreover from P2 have (μ j. j∈A∧ fj=x) ∈ A ∧ f(μ j. j∈A∧ fj=x)=x
        by (rule LeastI)
      ultimately have R: f(μ j. j∈A∧ fj=w)=w f(μ j. j∈A∧ fj=x)=x
        by auto
      from AS have (μ j. j∈A∧ f(j)=w)=(μ j. j∈A ∧ f(j)=x)
        using apply_equality fun2 by auto
      hence f(μ j. j∈A ∧ f(j)=w) = f(μ j. j∈A ∧ f(j)=x) by auto
      with R(1) have w = f(μ j. j∈A∧ fj=x) by auto
      with R(2) have w=x by auto
    }
    hence ∀w∈B. ∀x∈B. g(w) = g(x) ⟶ w = x
      by auto
    with fun2 show g∈inj(B,A) unfolding inj_def by auto
  qed
  then show thesis unfolding lepoll_def by auto
qed
```

The difference with the previous result is that in this one A is not a subset of an ordinal, it is only injective with one.

```
theorem surj_fun_inv_2:
  assumes f:surj(A,B) A≲Q Ord(Q)
  shows B≲A
proof-
  from assms(2) obtain h where h_def: h∈inj(A,Q) using lepoll_def by
auto
  then have bij: h∈bij(A,range(h)) using inj_bij_range by auto
  then obtain h1 where h1∈bij(range(h),A) using bij_converse_bij by
auto
  then have h1 ∈ surj(range(h),A) using bij_def by auto
  with assms(1) have (f O h1)∈surj(range(h),B) using comp_surj by auto
  moreover
  {
    fix x
    assume p: x∈range(h)
    from bij have h∈surj(A,range(h)) using bij_def by auto
    with p obtain q where q∈A and h(q)=x using surj_def by auto
```

```
    then have x∈Q using h_def inj_def by auto
  }
  then have range(h)⊆Q by auto
  ultimately have B≲range(h) using surj_fun_inv assms(3) by auto
  moreover have range(h)≈A using bij eqpoll_def eqpoll_sym by blast
  ultimately show B≲A using lepoll_eq_trans by auto
qed
```

17.4 Finite choice

In ZF every finite collection of non-empty sets has a choice function, i.e. a function that selects one element from each set of the collection. In this section we prove various forms of that claim.

The axiom of finite choice always holds.

```
theorem finite_choice:
  assumes n∈nat
  shows {the axiom of} n {choice holds}
proof -
  note assms(1)
  moreover
  {
    fix M N assume M≲0 ∀t∈M. Nt≠0
    then have M=0 using lepoll_0_is_0 by auto
    then have {⟨t,0⟩. t∈M}:Pi(M,λt. Nt) unfolding Pi_def domain_def function_def
Sigma_def by auto
    moreover from ‹M=0› have ∀t∈M. {⟨t,0⟩. t∈M}t∈Nt by auto
    ultimately have (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) by auto
  }
  then have (∀ M N. (M ≲0 ∧  (∀t∈M. Nt≠0)) ⟶ (∃f. f:Pi(M,λt. Nt)
∧ (∀t∈M. ft∈Nt)))
    by auto
  then have {the axiom of} 0 {choice holds} using AxiomCardinalChoiceGen_def
nat_into_Card
    by auto
  moreover {
    fix x
    assume as: x∈nat {the axiom of} x {choice holds}
    {
      fix M N assume ass: M≲succ(x) ∀t∈M. Nt≠0
      {
        assume M≲x
        from as(2) ass(2) have
          (M ≲ x ∧ (∀t∈M. N  t ≠ 0)) ⟶ (∃f. f ∈ Pi(M,λt. N  t) ∧
(∀t∈M. f  t ∈ N  t))
            unfolding AxiomCardinalChoiceGen_def by auto
        with ‹M≲x› ass(2) have (∃f. f ∈ Pi(M,λt. N  t) ∧ (∀t∈M. f
t ∈ N  t))
          by auto
```

```
      }
      moreover
      {
        assume M≈succ(x)
        then obtain f where f:f∈bij(succ(x),M) using eqpoll_sym eqpoll_def
by blast
        moreover
        have x∈succ(x) unfolding succ_def by auto
        ultimately have restrict(f,succ(x)-{x})∈bij(succ(x)-{x},M-{fx})
using bij_restrict_rem
          by auto
        moreover
        have x∉x using mem_not_refl by auto
        then have succ(x)-{x}=x unfolding succ_def by auto
        ultimately have restrict(f,x)∈bij(x,M-{fx}) by auto
        then have x≈M-{fx} unfolding eqpoll_def by auto
        then have M-{fx}≈x using eqpoll_sym by auto
        then have M-{fx}≲x using eqpoll_imp_lepoll by auto
        with as(2) ass(2) have (∃g. g ∈ Pi(M-{fx},λt. N  t) ∧ (∀t∈M-{fx}.
g  t ∈ N  t))
          unfolding AxiomCardinalChoiceGen_def by auto
        then obtain g where g: g∈ Pi(M-{fx},λt. N  t) ∀t∈M-{fx}. g
t ∈ N  t
          by auto
        from f have ff: fx∈M using bij_def inj_def apply_funtype by auto
        with ass(2) have N(fx)≠0 by auto
        then obtain y where y: y∈N(fx) by auto
        from g(1) have gg: g⊆Sigma(M-{fx},()(N)) unfolding Pi_def by
auto
        with y ff have g ∪{⟨fx, y⟩}⊆Sigma(M, ()(N)) unfolding Sigma_def
by auto
        moreover
        from g(1) have dom: M-{fx}⊆domain(g) unfolding Pi_def by auto
        then have M⊆domain(g ∪{⟨fx, y⟩}) unfolding domain_def by auto

        moreover
        from gg g(1) have noe: ~(∃t. ⟨fx,t⟩∈g) and function(g)
          unfolding domain_def Pi_def Sigma_def by auto
        with dom have fg: function(g ∪{⟨fx, y⟩}) unfolding function_def
by blast
        ultimately have PP: g ∪{⟨fx, y⟩}∈Pi(M,λt. N  t) unfolding Pi_def
by auto
        have ⟨fx, y⟩ ∈ g ∪{⟨fx, y⟩} by auto
        from this fg have (g ∪{⟨fx, y⟩})(fx)=y by (rule function_apply_equality)
        with y have (g ∪{⟨fx, y⟩})(fx)∈N(fx) by auto
        moreover
        {
          fix t assume A:t∈M-{fx}
          with g(1) have ⟨t,gt⟩∈g using apply_Pair by auto
```

```
        then have ⟨t,gt⟩∈(g ∪{⟨fx, y⟩}) by auto
        then have (g ∪{⟨fx, y⟩})t=gt using apply_equality PP by auto
        with A have (g ∪{⟨fx, y⟩})t∈Nt using g(2) by auto
      }
      ultimately have ∀t∈M. (g ∪{⟨fx, y⟩})t∈Nt by auto
      with PP have ∃g. g∈Pi(M,λt. N  t) ∧ (∀t∈M. gt∈Nt) by auto
    }
  ultimately have ∃g. g ∈ Pi(M, λt. Nt) ∧ (∀t∈M. g  t ∈ N  t) us-
ing as(1) ass(1)
    lepoll_succ_disj by auto
  }
  then have ∀M N. M ≲ succ(x)∧(∀t∈M. Nt≠0)⟶(∃g. g ∈ Pi(M,λt. N
t) ∧ (∀t∈M. g  t ∈ N  t))
    by auto
  then have {the axiom of}succ(x){choice holds}
    using AxiomCardinalChoiceGen_def nat_into_Card as(1) nat_succI by
auto
 }
 ultimately show thesis by (rule nat_induct)
qed
```

The choice functions of a collection $\mathcal{A}$ are functions f defined on $\mathcal{A}$ and valued in $\bigcup \mathcal{A}$ such that $f(A) \in A$ for every $A \in \mathcal{A}$.

```
definition
 ChoiceFunctions(𝒜) ≡ {f∈𝒜→⋃𝒜. ∀A∈𝒜. f(A)∈A}
```

For finite collections of non-empty sets the set of choice functions is non-empty.

```
theorem finite_choice1: assumes Finite(𝒜) and ∀A∈𝒜. A≠∅
 shows ChoiceFunctions(𝒜) ≠ ∅
proof -
 let N = id(𝒜)
 let 𝒩 = (λt. N(t))
 from assms(1) obtain n where n∈nat and 𝒜≈n
  unfolding Finite_def by auto
 from assms(2) ‹𝒜≈n› have 𝒜≲n and ∀A∈𝒜. N(A)≠∅
  using eqpoll_imp_lepoll by simp_all
 with ‹n∈nat› obtain f where f∈Pi(𝒜,𝒩)
  using finite_choice unfolding AxiomCardinalChoiceGen_def by blast
 have Pi(𝒜,𝒩) = {f∈𝒜→(⋃A∈𝒜. 𝒩(A)). ∀A∈𝒜. f(A)∈𝒩(A)}
  by (rule pi_fun_space)
 with ‹f∈Pi(𝒜,𝒩)› show thesis
  unfolding ChoiceFunctions_def by auto
qed
```

If a set X is finite and such that for every $x \in X$ we can find $y \in$ such that the property $P(x, y)$ holds, then there there is a function $f : X \to Y$ such that $P(x, f(x))$ holds for every $x \in X$.

```
lemma finite_choice_fun: assumes Finite(X) ∀x∈X. ∃y∈Y. P(x,y)
```

```
  shows ∃f∈X→Y. ∀x∈X. P(x,f(x))
proof -
  let N = {⟨x,{y∈Y. P(x,y)}⟩. x∈X}
  let 𝒩 = (λt. N(t))
  from assms(1) obtain n where n∈nat and X≈n
    unfolding Finite_def by auto
  have I: ∀x∈X. N(x) = {y∈Y. P(x,y)} using ZF_fun_from_tot_val2 by simp
  with assms(2) ‹X≈n› have X≲n and ∀x∈X. N(x)≠∅
    using eqpoll_imp_lepoll by auto
  with ‹n∈nat› obtain f where f∈Pi(X,𝒩) and II: ∀x∈X. f(x)∈N(x)
    using finite_choice unfolding AxiomCardinalChoiceGen_def by blast
  have Pi(X,𝒩) = {f∈X→(⋃x∈X. 𝒩(x)). ∀x∈X. f(x)∈𝒩(x)}
    by (rule pi_fun_space)
  with ‹f∈Pi(X,𝒩)› I II show thesis using func1_1_L1A by auto
qed

end
```

18 Finite choice and order relations

```
theory FinOrd_ZF_1 imports FinOrd_ZF Cardinal_ZF

begin
```

In this theory we continue the subject of finite sets and order relation from FinOrd_ZF with some consequences of finite choice for down-directed sets.

18.1 Finte choice and preorders

In the Order_ZF theory we define what it means that a relation r down-directs a set X: each two elements of X have a common lower bound in X. If the relation is a preorder (i.e. is reflexive and transitive) and it down-directs X we say that X is a down-directed set (by the relation r).

The next lemma states that each finite subset of a down-directed set has a lower bound in X.

```
lemma fin_dir_set_bounded:
  assumes IsDownDirectedSet(X,r) and B∈FinPow(X)
  shows ∃x∈X.∀t∈B. ⟨x,t⟩∈r
proof -
  from assms(1) have ∃x∈X.∀t∈∅. ⟨x,t⟩∈r
    unfolding IsDownDirectedSet_def DownDirects_def by auto
  moreover have
    ∀A∈FinPow(X). (∃x∈X.∀t∈A. ⟨x,t⟩∈r)⟶(∀a∈X. ∃m∈X.∀t∈A∪{a}. ⟨m,t⟩∈r)
  proof -
    { fix A assume A∈FinPow(X) and I: ∃x∈X.∀t∈A. ⟨x,t⟩∈r
      { fix a assume a∈X
```

```
      from I obtain x where x∈X and II: ∀t∈A. ⟨x,t⟩∈r by auto
      from assms(1) ‹a∈X› ‹x∈X› obtain m where
        m∈X ⟨m,a⟩∈r ⟨m,x⟩∈r
        unfolding IsDownDirectedSet_def DownDirects_def by auto
      with assms(1) II have ∃m∈X.∀t∈A∪{a}. ⟨m,t⟩∈r
        unfolding IsDownDirectedSet_def IsPreorder_def trans_def
          by blast
    } hence ∀a∈X. ∃x∈X.∀t∈A∪{a}. ⟨x,t⟩∈r by blast
  } thus thesis by simp
qed
moreover note assms(2)
ultimately show thesis by (rule FinPow_induct)
qed
```

Suppose Y is a set down-directed by a (preorder) relation r and f, g are funtions defined on two finite subsets A, B, resp., of X, valued in Y (i.e.$f : A \to Y$, $f : B \to Y$ and A, B are finite subsets of X). Then there exist a function $h : A \cup B \to Y$ that is a lower bound for f on A and for g on B.

```
lemma two_fun_low_bound:
  assumes IsDownDirectedSet(Y,r) A∈FinPow(X) B∈FinPow(X) f:A→Y g:B→Y
  shows ∃h∈A∪B→Y. (∀x∈A. ⟨h(x),f(x)⟩∈r) ∧ (∀x∈B. ⟨h(x),g(x)⟩∈r)
proof -
  from assms(2,3) have Finite(A∪B) using union_finpow
    unfolding FinPow_def by simp
  { fix x assume x∈A∪B
    from assms(4,5) have f{x}∪g{x} ∈ FinPow(Y)
      using image_singleton_fin union_finpow by simp
    with assms(1) have ∃y∈Y.∀t∈(f{x}∪g{x}). ⟨y,t⟩∈r
      using fin_dir_set_bounded by simp
  } hence ∀x∈A∪B. ∃y∈Y.∀t∈(f{x}∪g{x}). ⟨y,t⟩∈r by auto
  with ‹Finite(A∪B)› have
    ∃h∈A∪B→Y. ∀x∈A∪B. ∀t∈(f{x}∪g{x}). ⟨h(x),t⟩∈r
    by (rule finite_choice_fun)
  then obtain h where h∈A∪B→Y and
    ∀x∈A∪B. ∀t∈(f{x}∪g{x}). ⟨h(x),t⟩∈r
    by auto
  with assms(4,5) have
    ∀x∈A. ⟨h(x),f(x)⟩∈r and ∀x∈B. ⟨h(x),g(x)⟩∈r
    using func_imagedef by simp_all
  with ‹h∈A∪B→Y› show thesis by auto
qed

end
```

19 Equivalence relations

```
theory EquivClass1 imports ZF.EquivClass func_ZF ZF1
```

begin

In this theory file we extend the work on equivalence relations done in the standard Isabelle's EquivClass theory. That development is very good and all, but we really would prefer an approach contained within the a standard ZF set theory, without extensions specific to Isabelle. That is why this theory is written.

19.1 Congruent functions and projections on the quotient

Suppose we have a set X with a relation $r \subseteq X \times X$ and a function $f : X \to X$. The function f can be compatible (congruent) with r in the sense that if two elements x, y are related then the values $f(x), f(x)$ are also related. This is especially useful if r is an equivalence relation as it allows to "project" the function to the quotient space X/r (the set of equivalence classes of r) and create a new function F that satifies the formula $F([x]_r) = [f(x)]_r$. When f is congruent with respect to r such definition of the value of F on the equivalence class $[x]_r$ does not depend on which x we choose to represent the class. In this section we also consider binary operations that are congruent with respect to a relation. These are important in algebra - the congruency condition allows to project the operation to obtain the operation on the quotient space.

First we define the notion of function that maps equivalent elements to equivalent values. We use similar names as in the Isabelle's standard `EquivClass` theory to indicate the conceptual correspondence of the notions.

definition
```
  Congruent(r,f) ≡
  (∀x y. ⟨x,y⟩ ∈ r  ⟶ ⟨f(x),f(y)⟩ ∈ r)
```

Now we will define the projection of a function onto the quotient space. In standard math the equivalence class of x with respect to relation r is usually denoted $[x]_r$. Here we reuse notation $r\{x\}$ instead. This means the image of the set $\{x\}$ with respect to the relation, which, for equivalence relations is exactly its equivalence class if you think about it.

definition
```
  ProjFun(A,r,f) ≡
  {⟨c,⋃x∈c. r{f(x)}⟩. c ∈ (A//r)}
```

Elements of equivalence classes belong to the set.

lemma `EquivClass_1_L1:`
  assumes `A1: equiv(A,r)` **and** `A2: C ∈ A//r` **and** `A3: x∈C`
  shows `x∈A`
proof -
  from `A2` **have** `C ⊆ ⋃ (A//r)` **by** `auto`

```
  with A1 A3 show x∈A
    using Union_quotient by auto
qed
```

The image of a subset of X under projection is a subset of A/r.

```
lemma EquivClass_1_L1A:
  assumes A⊆X shows {r{x}. x∈A} ⊆ X//r
  using assms quotientI by auto
```

If an element belongs to an equivalence class, then its image under relation is this equivalence class.

```
lemma EquivClass_1_L2:
  assumes A1: equiv(A,r)  C ∈ A//r and A2: x∈C
  shows r{x} = C
proof -
  from A1 A2 have x ∈ r{x}
    using EquivClass_1_L1  equiv_class_self by simp
  with A2 have I: r{x}∩C ≠ 0 by auto
  from A1 A2 have r{x} ∈ A//r
    using EquivClass_1_L1 quotientI by simp
  with A1 I show thesis
    using quotient_disj by blast
qed
```

Elements that belong to the same equivalence class are equivalent.

```
lemma EquivClass_1_L2A:
  assumes equiv(A,r)  C ∈ A//r  x∈C  y∈C
  shows ⟨x,y⟩ ∈ r
  using assms EquivClass_1_L2 EquivClass_1_L1 equiv_class_eq_iff
  by simp
```

Elements that have the same image under an equivalence relation are equivalent. This is the same as `eq_equiv_class` from standard Isabelle/ZF's `EquivClass` theory, just copied here to be easier to find.

```
lemma same_image_equiv:
  assumes equiv(A,r) y∈A r{x} = r{y}
  shows ⟨x,y⟩ ∈ r using assms eq_equiv_class by simp
```

Every x is in the class of y, then they are equivalent.

```
lemma EquivClass_1_L2B:
  assumes A1: equiv(A,r) and A2: y∈A and A3: x ∈ r{y}
  shows ⟨x,y⟩ ∈ r
proof -
  from A2 have  r{y} ∈ A//r
    using quotientI by simp
  with A1 A3 show thesis using
    EquivClass_1_L1 equiv_class_self equiv_class_nondisjoint by blast
qed
```

If a function is congruent then the equivalence classes of the values that come from the arguments from the same class are the same.

```
lemma EquivClass_1_L3:
  assumes A1: equiv(A,r) and A2: Congruent(r,f)
  and A3: C ∈ A//r  x∈C  y∈C
  shows r{f(x)} = r{f(y)}
proof -
  from A1 A3 have ⟨x,y⟩ ∈ r
    using EquivClass_1_L2A by simp
  with A2 have  ⟨f(x),f(y)⟩ ∈ r
    using Congruent_def by simp
  with A1 show thesis using equiv_class_eq by simp
qed
```

The values of congruent functions are in the space.

```
lemma EquivClass_1_L4:
  assumes A1: equiv(A,r) and A2: C ∈ A//r  x∈C
  and A3: Congruent(r,f)
  shows f(x) ∈ A
proof -
  from A1 A2 have x∈A
    using EquivClass_1_L1 by simp
  with A1 have ⟨x,x⟩ ∈ r
    using equiv_def refl_def by simp
  with A3 have  ⟨f(x),f(x)⟩ ∈ r
    using Congruent_def by simp
  with A1 show thesis using equiv_type by auto
qed
```

Equivalence classes are not empty.

```
lemma EquivClass_1_L5:
  assumes A1: refl(A,r) and A2: C ∈ A//r
  shows C≠0
proof -
  from A2 obtain x where I: C = r{x} and x∈A
    using quotient_def by auto
  from A1 ‹x∈A› have x ∈ r{x} using refl_def by auto
  with I show thesis by auto
qed
```

To avoid using an axiom of choice, we define the projection using the expression $\bigcup_{x\in C} r(\{f(x)\})$. The next lemma shows that for congruent function this is in the quotient space A/r.

```
lemma EquivClass_1_L6:
  assumes A1: equiv(A,r) and A2: Congruent(r,f)
  and A3: C ∈ A//r
  shows (⋃x∈C. r{f(x)}) ∈ A//r
proof -
```

```
  from A1 have refl(A,r) unfolding equiv_def by simp
  with A3 have C≠0 using EquivClass_1_L5 by simp
  moreover from A2 A3 A1 have ∀x∈C. r{f(x)} ∈ A//r
    using EquivClass_1_L4 quotientI by auto
  moreover from A1 A2 A3 have
    ∀x y. x∈C ∧ y∈C ⟶ r{f(x)} = r{f(y)}
    using EquivClass_1_L3 by blast
  ultimately show thesis by (rule ZF1_1_L2)
qed
```

Congruent functions can be projected.

```
lemma EquivClass_1_T0:
  assumes equiv(A,r)  Congruent(r,f)
  shows ProjFun(A,r,f) : A//r → A//r
  using assms EquivClass_1_L6 ProjFun_def ZF_fun_from_total
  by simp
```

We now define congruent functions of two variables (binary funtions). The predicate `Congruent2` corresponds to `congruent2` in Isabelle's standard `EquivClass` theory, but uses ZF-functions rather than meta-functions.

```
definition
  Congruent2(r,f) ≡
  (∀x₁ x₂ y₁ y₂. ⟨x₁,x₂⟩ ∈ r ∧ ⟨y₁,y₂⟩ ∈ r  ⟶
  ⟨f⟨x₁,y₁⟩, f⟨x₂,y₂⟩ ⟩ ∈ r)
```

Next we define the notion of projecting a binary operation to the quotient space. This is a very important concept that allows to define quotient groups, among other things.

```
definition
  ProjFun2(A,r,f) ≡
  {⟨p,⋃ z ∈ fst(p)×snd(p). r{f(z)}⟩. p ∈ (A//r)×(A//r) }
```

The following lemma is a two-variables equivalent of `EquivClass_1_L3`.

```
lemma EquivClass_1_L7:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: C₁ ∈ A//r  C₂ ∈ A//r
  and A4: z₁ ∈ C₁×C₂  z₂ ∈ C₁×C₂
  shows r{f(z₁)} = r{f(z₂)}
proof -
  from A4 obtain x₁ y₁ x₂ y₂ where
    x₁∈C₁ and y₁∈C₂ and z₁ = ⟨x₁,y₁⟩ and
    x₂∈C₁ and y₂∈C₂ and z₂ = ⟨x₂,y₂⟩
    by auto
  with A1 A3 have ⟨x₁,x₂⟩ ∈ r and ⟨y₁,y₂⟩ ∈ r
    using EquivClass_1_L2A by auto
  with A2 have ⟨f⟨x₁,y₁⟩,f⟨x₂,y₂⟩⟩ ∈ r
    using Congruent2_def by simp
  with A1 ‹z₁ = ⟨x₁,y₁⟩› ‹z₂ = ⟨x₂,y₂⟩› show thesis
```

```
    using equiv_class_eq by simp
qed
```

The values of congruent functions of two variables are in the space.

```
lemma EquivClass_1_L8:
  assumes A1: equiv(A,r) and A2: C₁ ∈ A//r and A3: C₂ ∈ A//r
  and A4: z ∈ C₁×C₂ and A5: Congruent2(r,f)
  shows f(z) ∈ A
proof -
  from A4 obtain x y where x∈C₁ and y∈C₂ and z = ⟨x,y⟩
    by auto
  with A1 A2 A3 have x∈A and y∈A
    using EquivClass_1_L1 by auto
  with A1 A4 have ⟨x,x⟩ ∈ r and ⟨y,y⟩ ∈ r
    using equiv_def refl_def by auto
  with A5 have ⟨f⟨x,y⟩, f⟨x,y⟩ ⟩ ∈ r
    using Congruent2_def by simp
  with A1 ‹z = ⟨x,y⟩› show thesis using equiv_type by auto
qed
```

The values of congruent functions are in the space. Note that although this lemma is intended to be used with functions, we don't need to assume that f is a function.

```
lemma EquivClass_1_L8A:
  assumes A1: equiv(A,r) and A2: x∈A  y∈A
  and A3: Congruent2(r,f)
  shows f⟨x,y⟩ ∈ A
proof -
  from A1 A2 have r{x} ∈ A//r r{y} ∈ A//r
    ⟨x,y⟩ ∈ r{x}×r{y}
    using equiv_class_self quotientI by auto
  with A1 A3 show thesis using EquivClass_1_L8 by simp
qed
```

The following lemma is a two-variables equivalent of `EquivClass_1_L6`.

```
lemma EquivClass_1_L9:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: p ∈ (A//r)×(A//r)
  shows (⋃ z ∈ fst(p)×snd(p). r{f(z)}) ∈ A//r
proof -
  from A3 have fst(p) ∈ A//r and snd(p) ∈ A//r
    by auto
  with A1 A2 have
    I: ∀z ∈ fst(p)×snd(p). f(z) ∈ A
    using EquivClass_1_L8 by simp
  from A3 A1 have fst(p)×snd(p) ≠ 0
    using equiv_def EquivClass_1_L5 Sigma_empty_iff
    by auto
```

```
  moreover from A1 I have
    ∀z ∈ fst(p)×snd(p). r{f(z)} ∈ A//r
    using quotientI by simp
  moreover from A1 A2 ‹fst(p) ∈ A//r› ‹snd(p) ∈ A//r› have
    ∀z₁ z₂. z₁ ∈ fst(p)×snd(p) ∧ z₂ ∈ fst(p)×snd(p) ⟶
    r{f(z₁)} = r{f(z₂)}
    using EquivClass_1_L7 by blast
   ultimately show thesis by (rule ZF1_1_L2)
qed
```

Congruent functions of two variables can be projected.

```
theorem EquivClass_1_T1:
  assumes equiv(A,r)  Congruent2(r,f)
  shows ProjFun2(A,r,f) : (A//r)×(A//r) → A//r
  using assms EquivClass_1_L9 ProjFun2_def ZF_fun_from_total
  by simp
```

The projection diagram commutes. I wish I knew how to draw this diagram in LaTeX.

```
lemma EquivClass_1_L10:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: x∈A  y∈A
  shows ProjFun2(A,r,f)⟨r{x},r{y}⟩ = r{f⟨x,y⟩}
proof -
  from A3 A1 have r{x} × r{y} ≠ 0
    using quotientI equiv_def EquivClass_1_L5 Sigma_empty_iff
    by auto
  moreover have
    ∀z ∈ r{x}×r{y}.  r{f(z)} = r{f⟨x,y⟩}
  proof
    fix z assume A4: z ∈ r{x}×r{y}
    from A1 A3 have
      r{x} ∈ A//r r{y} ∈ A//r
      ⟨x,y⟩ ∈ r{x}×r{y}
      using quotientI equiv_class_self by auto
    with A1 A2 A4 show
      r{f(z)} = r{f⟨x,y⟩}
      using EquivClass_1_L7 by blast
  qed
  ultimately have
    (⋃z ∈ r{x}×r{y}. r{f(z)}) =  r{f⟨x,y⟩}
    by (rule ZF1_1_L1)
  moreover have
    ProjFun2(A,r,f)⟨r{x},r{y}⟩ = (⋃z ∈ r{x}×r{y}. r{f(z)})
    proof -
      from assms have
 ProjFun2(A,r,f) : (A//r)×(A//r) → A//r
 ⟨r{x},r{y}⟩ ∈ (A//r)×(A//r)
 using EquivClass_1_T1 quotientI by auto
```

```
      then show thesis using ProjFun2_def ZF_fun_from_tot_val
 by auto
    qed
  ultimately show thesis by simp
qed
```

19.2 Projecting commutative, associative and distributive operations.

In this section we show that if the operations are congruent with respect to an equivalence relation then the projection to the quotient space preserves commutativity, associativity and distributivity.

The projection of commutative operation is commutative.

```
lemma EquivClass_2_L1: assumes
  A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f {is commutative on} A
  and A4: c1 ∈ A//r  c2 ∈ A//r
  shows ProjFun2(A,r,f)⟨c1,c2⟩ = ProjFun2(A,r,f)⟨c2,c1⟩
proof -
  from A4 obtain x y where D1:
    c1 = r{x}  c2 = r{y}
    x∈A  y∈A
    using quotient_def by auto
  with A1 A2 have ProjFun2(A,r,f)⟨c1,c2⟩ = r{f⟨x,y⟩}
    using EquivClass_1_L10 by simp
  also from A3 D1 have
    r{f⟨x,y⟩} = r{f⟨y,x⟩}
    using IsCommutative_def by simp
  also from A1 A2 D1 have
    r{f⟨y,x⟩} = ProjFun2(A,r,f) ⟨c2,c1⟩
    using EquivClass_1_L10 by simp
  finally show thesis by simp
qed
```

The projection of commutative operation is commutative.

```
theorem EquivClass_2_T1:
  assumes equiv(A,r) and Congruent2(r,f)
  and f {is commutative on} A
  shows ProjFun2(A,r,f) {is commutative on} A//r
  using assms IsCommutative_def EquivClass_2_L1 by simp
```

The projection of an associative operation is associative.

```
lemma EquivClass_2_L2:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f {is associative on} A
  and A4: c1 ∈ A//r  c2 ∈ A//r  c3 ∈ A//r
  and A5: g = ProjFun2(A,r,f)
```

```
  shows g⟨g⟨c1,c2⟩,c3⟩ = g⟨c1,g⟨c2,c3⟩⟩
proof -
  from A4 obtain x y z where D1:
    c1 = r{x}  c2 = r{y}  c3 = r{z}
    x∈A  y∈A  z∈A
    using quotient_def by auto
  with A3 have T1:f⟨x,y⟩ ∈ A  f⟨y,z⟩ ∈ A
    using IsAssociative_def apply_type by auto
  with A1 A2 D1 A5 have
    g⟨g⟨c1,c2⟩,c3⟩ =  r{f⟨f⟨x,y⟩,z⟩}
    using EquivClass_1_L10 by simp
  also from D1 A3 have
    ... = r{f⟨x,f⟨y,z⟩ ⟩}
    using IsAssociative_def by simp
  also from T1 A1 A2 D1 A5 have
    ... = g⟨c1,g⟨c2,c3⟩⟩
    using EquivClass_1_L10 by simp
  finally show thesis by simp
qed
```

The projection of an associative operation is associative on the quotient.

```
theorem EquivClass_2_T2:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f {is associative on} A
  shows ProjFun2(A,r,f) {is associative on} A//r
proof -
  let g = ProjFun2(A,r,f)
  from A1 A2 have
    g ∈ (A//r)×(A//r) → A//r
    using EquivClass_1_T1 by simp
  moreover from A1 A2 A3 have
    ∀c1 ∈ A//r.∀c2 ∈ A//r.∀c3 ∈ A//r.
    g⟨g⟨c1,c2⟩,c3⟩ = g⟨c1,g⟨c2,c3⟩⟩
    using EquivClass_2_L2 by simp
  ultimately show thesis
    using IsAssociative_def by simp
qed
```

The essential condition to show that distributivity is preserved by projections to quotient spaces, provided both operations are congruent with respect to the equivalence relation.

```
lemma EquivClass_2_L3:
  assumes A1: IsDistributive(X,A,M)
  and A2: equiv(X,r)
  and A3: Congruent2(r,A) Congruent2(r,M)
  and A4: a ∈ X//r  b ∈ X//r  c ∈ X//r
  and A5: A_p = ProjFun2(X,r,A) M_p = ProjFun2(X,r,M)
  shows M_p⟨a,A_p⟨b,c⟩⟩ = A_p⟨ M_p⟨a,b⟩,M_p⟨a,c⟩⟩ ∧
  M_p⟨ A_p⟨b,c⟩,a ⟩ = A_p⟨ M_p⟨b,a⟩, M_p⟨c,a⟩⟩
```

```
proof
  from A4 obtain x y z where x∈X  y∈X  z∈X
    a = r{x}  b = r{y}  c = r{z}
    using quotient_def by auto
  with A1 A2 A3 A5 show
    Mp⟨a,Ap⟨b,c⟩⟩ = Ap⟨ Mp⟨a,b⟩,Mp⟨a,c⟩⟩ and
    Mp⟨ Ap⟨b,c⟩,a ⟩ = Ap⟨ Mp⟨b,a⟩, Mp⟨c,a⟩⟩
    using EquivClass_1_L8A EquivClass_1_L10 IsDistributive_def
    by auto
qed
```

Distributivity is preserved by projections to quotient spaces, provided both operations are congruent with respect to the equivalence relation.

```
lemma EquivClass_2_L4: assumes A1: IsDistributive(X,A,M)
  and A2: equiv(X,r)
  and A3: Congruent2(r,A) Congruent2(r,M)
  shows IsDistributive(X//r,ProjFun2(X,r,A),ProjFun2(X,r,M))
proof-
 let Ap = ProjFun2(X,r,A)
 let Mp = ProjFun2(X,r,M)
 from A1 A2 A3 have
   ∀a∈X//r.∀b∈X//r.∀c∈X//r.
   Mp⟨a,Ap⟨b,c⟩⟩ = Ap⟨Mp⟨a,b⟩,Mp⟨a,c⟩⟩ ∧
   Mp⟨Ap⟨b,c⟩,a⟩ = Ap⟨Mp⟨b,a⟩,Mp⟨c,a⟩⟩
   using EquivClass_2_L3 by simp
 then show thesis using IsDistributive_def by simp
qed
```

19.3 Saturated sets

In this section we consider sets that are saturated with respect to an equivalence relation. A set A is saturated with respect to a relation r if $A = r^{-1}(r(A))$. For equivalence relations saturated sets are unions of equivalence classes. This makes them useful as a tool to define subsets of the quotient space using properties of representants. Namely, we often define a set $B \subseteq X/r$ by saying that $[x]_r \in B$ iff $x \in A$. If A is a saturated set, this definition is consistent in the sense that it does not depend on the choice of x to represent $[x]_r$.

The following defines the notion of a saturated set. Recall that in Isabelle `r-(A)` is the inverse image of A with respect to relation r. This definition is not specific to equivalence relations.

```
definition
  IsSaturated(r,A) ≡ A = r-(r(A))
```

For equivalence relations a set is saturated iff it is an image of itself.

```
lemma EquivClass_3_L1: assumes A1: equiv(X,r)
```

```
  shows IsSaturated(r,A) ⟷ A = r(A)
proof
  assume IsSaturated(r,A)
  then have A = (converse(r) O r)(A)
    using IsSaturated_def vimage_def image_comp
    by simp
  also from A1 have ... = r(A)
    using equiv_comp_eq by simp
  finally show A = r(A) by simp
next assume A = r(A)
  with A1 have A = (converse(r) O r)(A)
    using equiv_comp_eq by simp
  also have ... =  r-(r(A))
    using vimage_def image_comp by simp
  finally have A =  r-(r(A)) by simp
  then show IsSaturated(r,A) using IsSaturated_def
    by simp
qed
```

For equivalence relations sets are contained in their images.

```
lemma EquivClass_3_L2: assumes A1: equiv(X,r) and A2: A⊆X
  shows A ⊆ r(A)
proof
  fix a assume a∈A
  with A1 A2 have a ∈ r{a}
    using equiv_class_self by auto
  with ‹a∈A› show a ∈ r(A) by auto
qed
```

The next lemma shows that if "$\sim$" is an equivalence relation and a set A is such that $a \in A$ and $a \sim b$ implies $b \in A$, then A is saturated with respect to the relation.

```
lemma EquivClass_3_L3: assumes A1: equiv(X,r)
  and A2: r ⊆ X×X and A3: A⊆X
  and A4: ∀x∈A. ∀y∈X. ⟨x,y⟩ ∈ r ⟶ y∈A
  shows IsSaturated(r,A)
proof -
  from A2 A4 have r(A) ⊆ A
    using image_iff by blast
  moreover from A1 A3 have A ⊆ r(A)
    using EquivClass_3_L2 by simp
  ultimately have A = r(A) by auto
  with A1 show IsSaturated(r,A) using EquivClass_3_L1
    by simp
qed
```

If $A \subseteq X$ and A is saturated and $x \sim y$, then $x \in A$ iff $y \in A$. Here we show only one direction.

```
lemma EquivClass_3_L4: assumes A1: equiv(X,r)
```

```
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: ⟨x,y⟩ ∈ r
  and A5: x∈X  y∈A
  shows x∈A
proof -
  from A1 A5 have x ∈ r{x}
    using equiv_class_self by simp
  with A1 A3 A4 A5 have x ∈ r(A)
    using equiv_class_eq equiv_class_self
    by auto
  with A1 A2 show x∈A
    using EquivClass_3_L1 by simp
qed
```

If $A \subseteq X$ and A is saturated and $x \sim y$, then $x \in A$ iff $y \in A$.

```
lemma EquivClass_3_L5: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: x∈X  y∈X
  and A5: ⟨x,y⟩ ∈ r
  shows x∈A ⟷ y∈A
proof
  assume y∈A
  with assms show x∈A using EquivClass_3_L4
    by simp
next assume x∈A
  from A1 A5 have ⟨y,x⟩ ∈ r
    using equiv_is_sym by blast
  with A1 A2 A3 A4 ‹x∈A› show y∈A
    using EquivClass_3_L4 by simp
qed
```

If A is saturated then $x \in A$ iff its class is in the projection of A.

```
lemma EquivClass_3_L6: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X and A4: x∈X
  and A5: B = {r{x}. x∈A}
  shows x∈A ⟷ r{x} ∈ B
proof
  assume x∈A
  with A5 show r{x} ∈ B by auto
next assume r{x} ∈ B
  with A5 obtain y where y ∈ A and r{x} = r{y}
    by auto
  with A1 A3 have ⟨x,y⟩ ∈ r
    using eq_equiv_class by auto
  with A1 A2 A3 A4  ‹y ∈ A› show x∈A
    using EquivClass_3_L4 by simp
qed
```

A technical lemma involving a projection of a saturated set and a logical epression with exclusive or. Note that we don't really care what `Xor` is here, this is true for any predicate.

```
lemma EquivClass_3_L7: assumes equiv(X,r)
  and IsSaturated(r,A) and A⊆X
  and x∈X  y∈X
  and B = {r{x}. x∈A}
  and (x∈A) Xor (y∈A)
  shows (r{x} ∈ B)  Xor (r{y} ∈ B)
  using assms EquivClass_3_L6 by simp
```

end

20 Finite sequences

theory `FiniteSeq_ZF` **imports** `Nat_ZF_IML func1`

begin

This theory treats finite sequences (i.e. maps $n \to X$, where $n = \{0, 1, .., n-1\}$ is a natural number) as lists. It defines and proves the properties of basic operations on lists: concatenation, appending and element etc.

20.1 Lists as finite sequences

A natural way of representing (finite) lists in set theory is through (finite) sequences. In such view a list of elements of a set X is a function that maps the set $\{0, 1, ..n-1\}$ into X. Since natural numbers in set theory are defined so that $n = \{0, 1, ..n-1\}$, a list of length n can be understood as an element of the function space $n \to X$.

We define the set of lists with values in set X as `Lists(X)`.

```
definition
  Lists(X) ≡ ⋃n∈nat.(n→X)
```

The set of nonempty X-value listst will be called `NELists(X)`.

```
definition
  NELists(X) ≡ ⋃n∈nat.(succ(n)→X)
```

We first define the shift that moves the second sequence to the domain $\{n, .., n + k - 1\}$, where n, k are the lengths of the first and the second sequence, resp. To understand the notation in the definitions below recall that in Isabelle/ZF `pred(n)` (predecessor of n is the previous natural number.

```
definition
  ShiftedSeq(b,n) ≡ {⟨j, b(j #- n)⟩. j ∈ NatInterval(n,domain(b))}
```

We define concatenation of two sequences as the union of the first sequence with the shifted second sequence. The result of concatenating lists a and b is called `Concat(a,b)`.

definition
```
  Concat(a,b) ≡ a ∪ ShiftedSeq(b,domain(a))
```

For a finite sequence we define the sequence of all elements except the first one. This corresponds to the "tail" function in Haskell. We call it `Tail` here as well.

definition
```
  Tail(a) ≡ {⟨k, a(succ(k))⟩. k ∈ pred(domain(a))}
```

A dual notion to `Tail` is the list of all elements of a list except the last one. Borrowing the terminology from Haskell again, we will call this `Init`.

definition
```
  Init(a) ≡ restrict(a,pred(domain(a)))
```

Another obvious operation we can talk about is appending an element at the end of a sequence. This is called `Append`.

definition
```
  Append(a,x) ≡ a ∪ {⟨domain(a),x⟩}
```

If lists are modeled as finite sequences (i.e. functions on natural intervals $\{0, 1, .., n-1\} = n$) it is easy to get the first element of a list as the value of the sequence at 0. The last element is the value at $n-1$. To hide this behind a familiar name we define the `Last` element of a list.

definition
```
  Last(a) ≡ a(pred(domain(a)))
```

A formula for tail of a finite list.

```
lemma tail_as_set: assumes n ∈ nat and a: n #+ 1 → X
  shows Tail(a) = {⟨k,a(k #+ 1)⟩. k∈n}
  using assms func1_1_L1 elem_nat_is_nat(2) succ_add_one(1)
  unfolding Tail_def by simp
```

Formula for the tail of a list defined by an expression:

```
lemma tail_formula: assumes n ∈ nat and ∀k ∈ n #+ 1. q(k) ∈ X
  shows Tail({⟨k,q(k)⟩. k ∈ n #+ 1}) = {⟨k,q(k #+ 1)⟩. k ∈ n}
proof -
  let a = {⟨k,q(k)⟩. k ∈ n #+ 1}
  from assms(2) have a : n #+ 1 → X
    by (rule ZF_fun_from_total)
  with assms(1) have Tail(a) = {⟨k,a(k #+ 1)⟩. k∈n}
    using tail_as_set by simp
  moreover have ∀k∈n. a(k #+ 1) = q(k #+ 1)
  proof -
```

```
    { fix k assume k∈n
      with assms(1) have k #+ 1 ∈ n #+ 1
        using succ_ineq1 elem_nat_is_nat(2) succ_add_one(1)
        by simp
      then have a(k #+ 1) = q(k #+ 1)
        by (rule ZF_fun_from_tot_val1)
    } thus thesis by simp
  qed
  ultimately show thesis by simp
qed
```

Codomain of a nonempty list is nonempty.

```
lemma nelist_vals_nonempty: assumes a:succ(n)→Y
  shows Y≠0 using assms codomain_nonempty by simp
```

Shifted sequence is a function on a the interval of natural numbers.

```
lemma shifted_seq_props:
  assumes A1: n ∈ nat  k ∈ nat and A2: b:k→X
  shows
  ShiftedSeq(b,n): NatInterval(n,k) → X
  ∀i ∈ NatInterval(n,k). ShiftedSeq(b,n)(i) = b(i #- n)
  ∀j∈k. ShiftedSeq(b,n)(n #+ j) = b(j)
proof -
  let I = NatInterval(n,domain(b))
  from A2 have Fact: I = NatInterval(n,k) using func1_1_L1 by simp
  with A1 A2 have ∀j∈ I. b(j #- n) ∈ X
    using inter_diff_in_len apply_funtype by simp
  then have
    {⟨j, b(j #- n)⟩. j ∈ I} : I → X by (rule ZF_fun_from_total)
  with Fact show thesis_1: ShiftedSeq(b,n): NatInterval(n,k) → X
    using ShiftedSeq_def by simp
  { fix i
    from Fact thesis_1 have  ShiftedSeq(b,n): I → X by simp
    moreover
    assume i ∈ NatInterval(n,k)
    with Fact have i ∈ I by simp
    moreover from Fact have
      ShiftedSeq(b,n) = {⟨i, b(i #- n)⟩. i ∈ I}
      using ShiftedSeq_def by simp
    ultimately have ShiftedSeq(b,n)(i) =  b(i #- n)
      by (rule ZF_fun_from_tot_val)
  } then show thesis1:
      ∀i ∈ NatInterval(n,k). ShiftedSeq(b,n)(i) = b(i #- n)
    by simp
  { fix j
    let i = n #+ j
    assume A3: j∈k
    with A1 have j ∈ nat using elem_nat_is_nat by blast
    then have i #- n = j using diff_add_inverse by simp
```

```
    with A3 thesis1 have ShiftedSeq(b,n)(i) = b(j)
      using NatInterval_def by auto
  } then show ∀j∈k. ShiftedSeq(b,n)(n #+ j) = b(j)
    by simp
qed
```

Basis properties of the contatenation of two finite sequences.

```
theorem concat_props:
  assumes A1: n ∈ nat  k ∈ nat and A2: a:n→X   b:k→X
  shows
  Concat(a,b): n #+ k → X
  ∀i∈n. Concat(a,b)(i) = a(i)
  ∀i ∈ NatInterval(n,k). Concat(a,b)(i) =  b(i #- n)
  ∀j ∈ k. Concat(a,b)(n #+ j) = b(j)
proof -
  from A1 A2 have
    a:n→X  and I: ShiftedSeq(b,n): NatInterval(n,k) → X
    and n ∩ NatInterval(n,k) = 0
    using shifted_seq_props length_start_decomp by auto
  then have
    a ∪ ShiftedSeq(b,n): n ∪ NatInterval(n,k) → X ∪ X
    by (rule fun_disjoint_Un)
  with A1 A2 show Concat(a,b): n #+ k → X
    using func1_1_L1 Concat_def length_start_decomp by auto
  { fix i assume i ∈ n
    with A1 I have i ∉ domain(ShiftedSeq(b,n))
      using length_start_decomp func1_1_L1 by auto
    with A2 have Concat(a,b)(i) = a(i)
      using func1_1_L1 fun_disjoint_apply1 Concat_def by simp
  } thus ∀i∈n. Concat(a,b)(i) = a(i) by simp
  { fix i assume A3: i ∈ NatInterval(n,k)
    with A1 A2 have i ∉ domain(a)
      using length_start_decomp func1_1_L1 by auto
    with A1 A2 A3 have Concat(a,b)(i) =  b(i #- n)
      using func1_1_L1 fun_disjoint_apply2 Concat_def shifted_seq_props
      by simp
  } thus II: ∀i ∈ NatInterval(n,k). Concat(a,b)(i) =  b(i #- n)
    by simp
  { fix j
    let i = n #+ j
    assume A3: j∈k
    with A1 have j ∈ nat using elem_nat_is_nat by blast
    then have i #- n = j using diff_add_inverse by simp
     with A3 II have Concat(a,b)(i) = b(j)
      using NatInterval_def by auto
  } thus ∀j ∈ k. Concat(a,b)(n #+ j) = b(j)
    by simp
qed
```

Properties of concatenating three lists.

```
lemma concat_concat_list:
  assumes A1: n ∈ nat  k ∈ nat  m ∈ nat and
  A2: a:n→X   b:k→X  c:m→X and
  A3: d = Concat(Concat(a,b),c)
  shows
  d : n #+ k #+ m → X
  ∀j ∈ n. d(j) = a(j)
  ∀j ∈ k. d(n #+ j) = b(j)
  ∀j ∈ m. d(n #+ k #+ j) = c(j)
proof -
  from A1 A2 have I:
    n #+ k ∈ nat   m ∈ nat
    Concat(a,b): n #+ k → X   c:m→X
    using concat_props by auto
  with A3 show d: n #+k #+ m → X
    using concat_props by simp
  from I have II: ∀i ∈ n #+ k.
    Concat(Concat(a,b),c)(i) = Concat(a,b)(i)
    by (rule concat_props)
  { fix j assume A4: j ∈ n
    moreover from A1 have n ⊆ n #+ k using add_nat_le by simp
    ultimately have j ∈ n #+ k by auto
    with A3 II have d(j) =  Concat(a,b)(j) by simp
    with A1 A2 A4 have d(j) = a(j)
      using concat_props by simp
  } thus ∀j ∈ n. d(j) = a(j) by simp
  { fix j assume A5: j ∈ k
    with A1 A3 II have d(n #+ j) = Concat(a,b)(n #+ j)
      using add_lt_mono by simp
    also from A1 A2 A5 have ... = b(j)
      using concat_props by simp
    finally have d(n #+ j) = b(j) by simp
  } thus ∀j ∈ k. d(n #+ j) = b(j) by simp
  from I have ∀j ∈ m. Concat(Concat(a,b),c)(n #+ k #+ j) = c(j)
    by (rule concat_props)
  with A3 show ∀j ∈ m. d(n #+ k #+ j) = c(j)
    by simp
qed
```

Properties of concatenating a list with a concatenation of two other lists.

```
lemma concat_list_concat:
  assumes A1: n ∈ nat  k ∈ nat  m ∈ nat and
  A2: a:n→X   b:k→X  c:m→X and
  A3: e = Concat(a, Concat(b,c))
  shows
  e : n #+ k #+ m → X
  ∀j ∈ n. e(j) = a(j)
  ∀j ∈ k. e(n #+ j) = b(j)
  ∀j ∈ m. e(n #+ k #+ j) = c(j)
```

```
proof -
  from A1 A2 have I:
    n ∈ nat  k #+ m ∈ nat
    a:n→X  Concat(b,c): k #+ m → X
    using concat_props by auto
  with A3 show  e : n #+k #+ m → X
    using concat_props add_assoc by simp
  from I have ∀j ∈ n. Concat(a, Concat(b,c))(j) = a(j)
    by (rule concat_props)
  with A3 show ∀j ∈ n. e(j) = a(j) by simp
  from I have II:
    ∀j ∈ k #+ m. Concat(a, Concat(b,c))(n #+ j) = Concat(b,c)(j)
    by (rule concat_props)
  { fix j assume A4: j ∈ k
    moreover from A1 have k ⊆ k #+ m using add_nat_le by simp
    ultimately have j ∈ k #+ m by auto
    with A3 II have e(n #+ j) =  Concat(b,c)(j) by simp
    also from A1 A2 A4 have ... = b(j)
      using concat_props by simp
    finally have e(n #+ j) = b(j) by simp
  } thus ∀j ∈ k. e(n #+ j) = b(j) by simp
  { fix j assume A5: j ∈ m
    with A1 II A3 have e(n #+ k #+ j) = Concat(b,c)(k #+ j)
      using add_lt_mono add_assoc by simp
    also from A1 A2 A5 have ... = c(j)
      using concat_props by simp
    finally have e(n #+ k #+ j) = c(j) by simp
  } then show ∀j ∈ m. e(n #+ k #+ j) = c(j)
    by simp
qed
```

Concatenation is associative.

```
theorem concat_assoc:
  assumes A1: n ∈ nat  k ∈ nat  m ∈ nat and
  A2: a:n→X   b:k→X   c:m→X
  shows Concat(Concat(a,b),c) =  Concat(a, Concat(b,c))
proof -
  let d = Concat(Concat(a,b),c)
  let e = Concat(a, Concat(b,c))
  from A1 A2 have
    d : n #+k #+ m → X and e : n #+k #+ m → X
    using concat_concat_list concat_list_concat by auto
  moreover have ∀i ∈  n #+k #+ m. d(i) = e(i)
  proof -
    { fix i assume i ∈ n #+k #+ m
      moreover from A1 have
 n #+k #+ m = n ∪ NatInterval(n,k) ∪ NatInterval(n #+ k,m)
 using adjacent_intervals3 by simp
      ultimately have
```

```
 i ∈ n ∨ i ∈ NatInterval(n,k) ∨ i ∈ NatInterval(n #+ k,m)
 by simp
      moreover
      { assume i ∈ n
 with A1 A2 have d(i) = e(i)
 using concat_concat_list concat_list_concat by simp }
      moreover
      { assume i ∈ NatInterval(n,k)
 then obtain j where j∈k and i = n #+ j
   using NatInterval_def by auto
 with A1 A2 have d(i) = e(i)
   using concat_concat_list concat_list_concat by simp }
      moreover
      { assume i ∈ NatInterval(n #+ k,m)
 then obtain j where j ∈ m and i = n #+ k #+ j
   using NatInterval_def by auto
 with A1 A2 have d(i) = e(i)
   using concat_concat_list concat_list_concat by simp }
      ultimately have d(i) = e(i) by auto
    } thus thesis by simp
  qed
  ultimately show d = e by (rule func_eq)
qed
```

Properties of `Tail`.

```
theorem tail_props:
  assumes A1: n ∈ nat and A2: a: succ(n) → X
  shows
  Tail(a) : n → X
  ∀k ∈ n. Tail(a)(k) = a(succ(k))
proof -
  from A1 A2 have ∀k ∈ n. a(succ(k)) ∈ X
    using succ_ineq apply_funtype by simp
  then have {⟨k, a(succ(k))⟩. k ∈ n} : n → X
    by (rule ZF_fun_from_total)
  with A2 show I: Tail(a) : n → X
    using func1_1_L1 pred_succ_eq Tail_def by simp
  moreover from A2 have Tail(a) = {⟨k, a(succ(k))⟩. k ∈ n}
    using func1_1_L1 pred_succ_eq Tail_def by simp
  ultimately show ∀k ∈ n. Tail(a)(k) = a(succ(k))
    by (rule ZF_fun_from_tot_val0)
qed
```

Essentially the second assertion of `tail_props` but formulated using notation $n + 1$ instead of `succ(n)`:

```
lemma tail_props2: assumes n ∈ nat a: n #+ 1 → X k∈n
  shows Tail(a)(k) = a(k #+ 1)
  using assms succ_add_one(1) tail_props(2) elem_nat_is_nat(2)
  by simp
```

A nonempty list can be decomposed into concatenation of its first element and the tail.

```
lemma first_concat_tail: assumes n∈nat a:succ(n)→X
  shows a = Concat({⟨0,a(0)⟩},Tail(a))
proof -
  let b = Concat({⟨0,a(0)⟩},Tail(a))
  have b:succ(n)→X and ∀k∈succ(n). a(k) = b(k)
  proof -
    from assms(1) have 0∈succ(n) using empty_in_every_succ by simp
    with assms(2) have a(0) ∈ X using apply_funtype by simp
    then have I: {⟨0,a(0)⟩}:{0}→X using pair_func_singleton by simp
    have {0}∈nat using one_is_nat by simp
    from assms have Tail(a): n → X using tail_props(1) by simp
    with assms(1) ‹{0}∈nat› I have b:{0} #+ n → X
      using concat_props(1) by simp
    with assms(1) show b:succ(n) → X using succ_add_one(3) by simp
    { fix k assume k∈succ(n)
      { assume k=0
        with assms(1) ‹{0}∈nat› I ‹Tail(a): n → X›
        have a(k) = b(k) using concat_props(2) pair_val
          by simp
      }
      moreover
      { assume k≠0
        from assms(1) ‹k∈succ(n)› have k∈nat
          using elem_nat_is_nat(2) by blast
        with ‹k≠0› obtain m where m∈nat and k=succ(m)
          using Nat_ZF_1_L3 by blast
        with assms(1) ‹k∈succ(n)› have m∈n using succ_mem
          by simp
        with ‹{0}∈nat› assms(1) I ‹Tail(a): n → X›
        have b({0} #+ m) = Tail(a)(m)
          using concat_props(4) by simp
        with assms ‹m∈nat› ‹k∈succ(n)› ‹k=succ(m)› ‹m∈n›
        have a(k) = b(k)
          using succ_add_one(3) tail_props(2) by simp
      }
      ultimately have a(k) = b(k) by blast
    } thus ∀k∈succ(n). a(k) = b(k) by simp
  qed
  with assms(2) show thesis by (rule func_eq)
qed
```

Properties of `Append`. It is a bit surprising that the we don't need to assume that n is a natural number.

```
theorem append_props:
  assumes A1: a: n → X and A2: x∈X and A3: b = Append(a,x)
  shows
  b : succ(n) → X
```

```
  ∀k∈n. b(k) = a(k)
  b(n) = x
proof -
  note A1
  moreover have I: n ∉ n using mem_not_refl by simp
  moreover from A1 A3 have II: b = a ∪ {⟨n,x⟩}
    using func1_1_L1 Append_def by simp
  ultimately have b : n ∪ {n} → X ∪ {x}
    by (rule func1_1_L11D)
  with A2 show b : succ(n) → X
    using succ_explained set_elem_add by simp
  from A1 I II show ∀k∈n. b(k) = a(k) and b(n) = x
    using func1_1_L11D by auto
qed
```

A special case of append_props: appending to a nonempty list does not change the head (first element) of the list.

```
corollary head_of_append:
  assumes n∈ nat and a: succ(n) → X and x∈X
  shows Append(a,x)(0) = a(0)
  using assms append_props empty_in_every_succ by auto
```

Tail commutes with Append.

```
theorem tail_append_commute:
  assumes A1: n ∈ nat and A2: a: succ(n) → X and A3: x∈X
  shows Append(Tail(a),x) = Tail(Append(a,x))
proof -
  let b = Append(Tail(a),x)
  let c = Tail(Append(a,x))
  from A1 A2 have I: Tail(a) : n → X using tail_props
    by simp
  from A1 A2 A3 have
    succ(n) ∈ nat and Append(a,x) : succ(succ(n)) → X
    using append_props by auto
  then have II: ∀k ∈ succ(n). c(k) = Append(a,x)(succ(k))
    by (rule tail_props)
  from assms have
    b : succ(n) → X and c : succ(n) → X
    using tail_props append_props by auto
  moreover have ∀k ∈ succ(n). b(k) = c(k)
  proof -
    { fix k assume k ∈ succ(n)
      hence k ∈ n ∨ k = n by auto
      moreover
      { assume A4: k ∈ n
 with assms II have c(k) = a(succ(k))
   using succ_ineq append_props by simp
 moreover
 from A3 I have ∀k∈n. b(k) = Tail(a)(k)
```

```
    using append_props by simp
  with A1 A2 A4 have b(k) =  a(succ(k))
    using tail_props by simp
  ultimately have b(k) = c(k) by simp }
      moreover
      { assume A5: k = n
  with A2 A3 I II have b(k) = c(k)
    using append_props by auto }
      ultimately have b(k) = c(k) by auto
    } thus thesis by simp
  qed
  ultimately show b = c by (rule func_eq)
qed
```

NELists are non-empty lists

```
lemma non_zero_List_func_is_NEList:
  shows NELists(X) = {a∈Lists(X). a≠0}
proof-
  { fix a assume as: a∈{a∈Lists(X). a≠0}
    from as obtain n where a: n∈nat a:n→ X unfolding Lists_def
      by auto
    { assume n=0
      with a(2) have a=0 unfolding Pi_def by auto
      with as have False by auto
    }
    hence n≠0 by auto
    with a(1) obtain k where k∈nat n=succ(k) using Nat_ZF_1_L3
      by auto
    with a(2) have a ∈ NELists(X) unfolding NELists_def by auto
  } moreover
  { fix a assume as: a∈NELists(X)
    then obtain k where k: a:succ(k)→X k∈nat
      unfolding NELists_def by auto
    { assume a=0
      hence domain(a) = 0 by auto
      with k(1) have succ(k) = 0 using domain_of_fun by auto
      hence False by auto
    } moreover
    { from k(2) have succ(k)∈nat using nat_succI by auto
      with k(1) have a∈Lists(X) unfolding Lists_def by auto
    } ultimately
    have a∈{a∈Lists(X). a≠0} by auto
  }
  ultimately show thesis by auto
qed
```

Properties of Init.

```
theorem init_props:
  assumes A1: n ∈ nat and A2: a: succ(n) → X
```

```
  shows
  Init(a) : n → X
  ∀k∈n. Init(a)(k) = a(k)
  a = Append(Init(a), a(n))
proof -
  have n ⊆ succ(n) by auto
  with A2 have restrict(a,n): n → X
    using restrict_type2 by simp
  moreover from A1 A2 have I: restrict(a,n) = Init(a)
    using func1_1_L1 pred_succ_eq Init_def by simp
  ultimately show thesis1: Init(a) : n → X by simp
  { fix k assume k∈n
    then have restrict(a,n)(k) = a(k)
      using restrict by simp
    with I have Init(a)(k) = a(k) by simp
  } then show thesis2: ∀k∈n. Init(a)(k) = a(k) by simp
  let b = Append(Init(a), a(n))
  from A2 thesis1 have II:
    Init(a) : n → X   a(n) ∈ X
    b = Append(Init(a), a(n))
    using apply_funtype by auto
  note A2
  moreover from II have b : succ(n) → X
    by (rule append_props)
  moreover have ∀k ∈ succ(n). a(k) = b(k)
  proof -
    { fix k assume A3: k ∈ n
      from II have ∀j∈n. b(j) = Init(a)(j)
 by (rule append_props)
      with thesis2 A3 have a(k) = b(k) by simp }
    moreover
    from II have b(n) = a(n)
      by (rule append_props)
    hence  a(n) = b(n) by simp
    ultimately show ∀k ∈ succ(n). a(k) = b(k)
      by simp
  qed
  ultimately show a = b by (rule func_eq)
qed
```

The initial part of a non-empty list is a list, and the domain of the original list is the successor of its initial part.

```
theorem init_NElist:
  assumes a ∈ NELists(X)
  shows Init(a) ∈ Lists(X) and succ(domain(Init(a))) = domain(a)
proof -
  from assms obtain n where n: n∈nat a:succ(n) → X
    unfolding NELists_def by auto
  then have tailF: Init(a):n → X using init_props(1) by auto
```

```
  with n(1) show Init(a) ∈ Lists(X) unfolding Lists_def by auto
  from tailF have domain(Init(a)) = n using domain_of_fun by auto
  moreover from n(2) have domain(a) = succ(n) using domain_of_fun
    by auto
  ultimately show succ(domain(Init(a))) = domain(a) by auto
qed
```

If we take init of the result of append, we get back the same list.

```
lemma init_append: assumes A1: n ∈ nat and A2: a:n→X and A3: x ∈ X
  shows Init(Append(a,x)) = a
proof -
  from A2 A3 have Append(a,x): succ(n)→X using append_props by simp
  with A1 have Init(Append(a,x)):n→X and ∀k∈n. Init(Append(a,x))(k)
= Append(a,x)(k)
    using init_props by auto
  with A2 A3 have ∀k∈n. Init(Append(a,x))(k) = a(k) using append_props
by simp
  with ‹Init(Append(a,x)):n→X› A2 show thesis by (rule func_eq)
qed
```

A reformulation of definition of `Init`.

```
lemma init_def: assumes n ∈ nat and a:succ(n)→X
  shows Init(a) = restrict(a,n)
  using assms func1_1_L1 Init_def by simp
```

Another reformulation of the definition of `Init`, starting with the expression defining the list.

```
lemma init_def_alt: assumes n∈nat and ∀k∈n #+ 1. q(k) ∈ X
  shows Init({⟨k,q(k)⟩. k∈n #+ 1}) = {⟨k,q(k)⟩. k∈n}
proof -
  let a = {⟨k,q(k)⟩. k∈n #+ 1}
  from assms(2) have a:n #+ 1→X by (rule ZF_fun_from_total)
  moreover from assms(1) have n #+ 1 = succ(n) using succ_add_one(1)
    by simp
  ultimately have a:succ(n)→X by simp
  with assms(1) have Init(a) = restrict(a,n) using init_def by simp
  moreover
  from assms(1) have n ⊆ n #+ 1 by auto
  then have restrict(a,n) = {⟨k,q(k)⟩. k∈n}
    by (rule restrict_def_alt)
  ultimately show thesis by simp
qed
```

A lemma about extending a finite sequence by one more value. This is just a more explicit version of `append_props`.

```
lemma finseq_extend:
  assumes  a:n→X   y∈X   b = a ∪ {⟨n,y⟩}
  shows
```

```
  b: succ(n) → X
  ∀k∈n. b(k) = a(k)
  b(n) = y
  using assms Append_def func1_1_L1 append_props by auto
```

The next lemma is a bit displaced as it is mainly about finite sets. It is proven here because it uses the notion of `Append`. Suppose we have a list of element of A is a bijection. Then for every element that does not belong to A we can we can construct a bijection for the set $A \cup \{x\}$ by appending x. This is just a specialised version of lemma `bij_extend_point` from `func1.thy`.

```
lemma bij_append_point:
  assumes A1: n ∈ nat and A2: b ∈ bij(n,X) and A3: x ∉ X
  shows Append(b,x) ∈ bij(succ(n), X ∪ {x})
proof -
  from A2 A3 have b ∪ {⟨n,x⟩} ∈ bij(n ∪ {n},X ∪ {x})
    using mem_not_refl bij_extend_point by simp
  moreover have Append(b,x) = b ∪ {⟨n,x⟩}
  proof -
    from A2 have b:n→X
      using bij_def surj_def by simp
    then have b : n → X ∪ {x} using func1_1_L1B
      by blast
    then show Append(b,x) = b ∪ {⟨n,x⟩}
      using Append_def func1_1_L1 by simp
  qed
  ultimately show thesis using succ_explained by auto
qed
```

The next lemma rephrases the definition of `Last`. Recall that in ZF we have $\{0, 1, 2, .., n\} = n + 1 =$succ$(n)$.

```
lemma last_seq_elem: assumes a: succ(n) → X shows Last(a) = a(n)
  using assms func1_1_L1 pred_succ_eq Last_def by simp
```

The last element of a non-empty list valued in X is in X.

```
lemma last_type: assumes a ∈ NELists(X) shows Last(a) ∈ X
  using assms last_seq_elem apply_funtype unfolding NELists_def
  by auto
```

The last element of a list of length at least 2 is the same as the last element of the tail of that list.

```
lemma last_tail_last: assumes n∈nat a: succ(succ(n)) → X
  shows Last(Tail(a)) = Last(a)
proof -
  from assms have Last(Tail(a)) = Tail(a)(n)
    using tail_props(1) last_seq_elem by blast
  also from assms have Tail(a)(n) = a(succ(n)) using tail_props(2)
    by blast
  also from assms(2) have a(succ(n)) = Last(a) using last_seq_elem
```

```
    by simp
  finally show thesis by simp
qed
```

If two finite sequences are the same when restricted to domain one shorter than the original and have the same value on the last element, then they are equal.

```
lemma finseq_restr_eq: assumes A1: n ∈ nat and
  A2: a: succ(n) → X  b: succ(n) → X and
  A3: restrict(a,n) = restrict(b,n) and
  A4: a(n) = b(n)
  shows a = b
proof -
  { fix k assume k ∈ succ(n)
    then have k ∈ n ∨ k = n by auto
    moreover
    { assume k ∈ n
      then have
 restrict(a,n)(k) = a(k) and restrict(b,n)(k) = b(k)
 using restrict by auto
      with A3 have a(k) = b(k) by simp }
    moreover
    { assume k = n
      with A4 have a(k) = b(k) by simp }
    ultimately have a(k) = b(k) by auto
  } then have ∀ k ∈ succ(n). a(k) = b(k) by simp
  with A2 show a = b by (rule func_eq)
qed
```

Concatenating a list of length 1 is the same as appending its first (and only) element. Recall that in ZF set theory $1 = \{0\}$.

```
lemma append_1elem: assumes A1: n ∈ nat and
  A2: a: n → X  and A3: b : 1 → X
  shows Concat(a,b) = Append(a,b(0))
proof -
  let C = Concat(a,b)
  let A = Append(a,b(0))
  from A1 A2 A3 have I:
    n ∈ nat  1 ∈ nat
    a:n→X   b:1→X by auto
  have C : succ(n) → X
  proof -
    from I have C : n #+ 1 → X
      by (rule concat_props)
    with A1 show C : succ(n) → X by simp
  qed
  moreover from A2 A3 have A : succ(n) → X
    using apply_funtype append_props by simp
  moreover have ∀k ∈ succ(n). C(k) = A(k)
```

```
  proof
    fix k assume k ∈ succ(n)
    moreover
    { assume k ∈ n
      moreover from I have ∀i ∈ n. C(i) = a(i)
 by (rule concat_props)
      moreover from A2 A3 have ∀i∈n. A(i) = a(i)
 using apply_funtype append_props by simp
      ultimately have C(k) =  A(k) by simp }
    moreover have C(n) = A(n)
    proof -
      from I have ∀j ∈ 1. C(n #+ j) = b(j)
 by (rule concat_props)
      with A1 A2 A3 show C(n) = A(n)
 using apply_funtype append_props by simp
    qed
    ultimately show C(k) = A(k) by auto
  qed
  ultimately show C = A by (rule func_eq)
qed
```

If $x \in X$ then the singleton set with the pair $\langle 0, x\rangle$ as the only element is a list of length 1 and hence a nonempty list.

```
lemma list_len1_singleton: assumes x∈X
  shows {⟨0,x⟩} : 1 → X and {⟨0,x⟩} ∈ NELists(X)
proof -
  from assms have {⟨0,x⟩} : {0} → X using pair_func_singleton
    by simp
  moreover have {0} = 1 by auto
  ultimately show {⟨0,x⟩} : 1 → X and {⟨0,x⟩} ∈ NELists(X)
    unfolding NELists_def by auto
qed
```

A singleton list is in fact a singleton set with a pair as the only element.

```
lemma list_singleton_pair: assumes A1: x:1→X shows x = {⟨0,x(0)⟩}
proof -
  from A1 have x = {⟨t,x(t)⟩. t∈1} by (rule fun_is_set_of_pairs)
  hence x = {⟨t,x(t)⟩. t∈{0} } by simp
  thus thesis by simp
qed
```

When we append an element to the empty list we get a list with length 1.

```
lemma empty_append1: assumes A1: x∈X
  shows Append(0,x): 1 → X and Append(0,x)(0) = x
proof -
  let a = Append(0,x)
  have a = {⟨0,x⟩} using Append_def by auto
  with A1 show a : 1 → X and a(0) = x
    using list_len1_singleton pair_func_singleton
```

```
    by auto
qed
```

Appending an element is the same as concatenating with certain pair.

```
lemma append_concat_pair:
  assumes n ∈ nat and a: n → X and x∈X
  shows Append(a,x) = Concat(a,{⟨0,x⟩})
  using assms list_len1_singleton append_1elem pair_val
  by simp
```

An associativity property involving concatenation and appending. For proof we just convert appending to concatenation and use `concat_assoc`.

```
lemma concat_append_assoc: assumes A1: n ∈ nat  k ∈ nat and
  A2: a:n→X   b:k→X and A3: x ∈ X
  shows Append(Concat(a,b),x) = Concat(a, Append(b,x))
proof -
  from A1 A2 A3 have
    n #+ k ∈ nat   Concat(a,b) : n #+ k → X   x ∈ X
    using concat_props by auto
  then have
    Append(Concat(a,b),x) =  Concat(Concat(a,b),{⟨0,x⟩})
    by (rule append_concat_pair)
  moreover
  from A1 A2 A3 have
    n ∈ nat  k ∈ nat  1 ∈ nat
     a:n→X   b:k→X  {⟨0,x⟩} :  1 → X
    using list_len1_singleton by auto
  then have
    Concat(Concat(a,b),{⟨0,x⟩}) = Concat(a, Concat(b,{⟨0,x⟩}))
    by (rule concat_assoc)
  moreover from A1 A2 A3 have Concat(b,{⟨0,x⟩}) =  Append(b,x)
    using list_len1_singleton append_1elem pair_val by simp
  ultimately show Append(Concat(a,b),x) = Concat(a, Append(b,x))
    by simp
qed
```

An identity involving concatenating with init and appending the last element.

```
lemma concat_init_last_elem:
  assumes n ∈ nat  k ∈ nat and
  a: n → X  and b : succ(k) → X
  shows Append(Concat(a,Init(b)),b(k)) = Concat(a,b)
  using assms init_props apply_funtype concat_append_assoc
  by simp
```

A lemma about creating lists by composition and how `Append` behaves in such case.

```
lemma list_compose_append:
```

```
  assumes A1: n ∈ nat and A2: a : n → X and
  A3: x ∈ X and A4: c : X → Y
  shows
  c O Append(a,x) : succ(n) → Y
  c O Append(a,x) = Append(c O a, c(x))
proof -
  let b = Append(a,x)
  let d = Append(c O a, c(x))
  from A2 A4 have c O a : n → Y
    using comp_fun by simp
  from A2 A3 have b : succ(n) → X
    using append_props by simp
  with A4 show c O b : succ(n) → Y
    using comp_fun by simp
  moreover from A3 A4 ‹c O a : n → Y› have
    d: succ(n) → Y
    using apply_funtype append_props by simp
  moreover have ∀k ∈ succ(n). (c O b) (k) = d(k)
  proof -
    { fix k assume k ∈ succ(n)
      with ‹b : succ(n) → X› have
 (c O b) (k) = c(b(k))
 using comp_fun_apply by simp
      with A2 A3 A4 ‹c O a : n → Y› ‹c O a : n → Y› ‹k ∈ succ(n)›
      have (c O b) (k) = d(k)
 using append_props comp_fun_apply apply_funtype
 by auto
    } thus thesis by simp
  qed
  ultimately show c O b = d by (rule func_eq)
qed
```

A lemma about appending an element to a list defined by set comprehension.

```
lemma set_list_append: assumes
  A1: ∀i ∈ succ(k). b(i) ∈ X and
  A2: a = {⟨i,b(i)⟩. i ∈ succ(k)}
  shows
  a: succ(k) → X
  {⟨i,b(i)⟩. i ∈ k}: k → X
  a = Append({⟨i,b(i)⟩. i ∈ k},b(k))
proof -
  from A1 have {⟨i,b(i)⟩. i ∈ succ(k)} : succ(k) → X
    by (rule ZF_fun_from_total)
  with A2 show a: succ(k) → X by simp
  from A1 have ∀i ∈ k. b(i) ∈ X
    by simp
  then show {⟨i,b(i)⟩. i ∈ k}: k → X
    by (rule ZF_fun_from_total)
  with A2 show a = Append({⟨i,b(i)⟩. i ∈ k},b(k))
```

```
    using func1_1_L1 Append_def by auto
qed
```

A version of `set_list_append` using $n + 1$ instead of `succ(n)`.

```
lemma set_list_append1:
  assumes n∈nat and ∀k∈n #+ 1. q(k) ∈ X
  defines a≡{⟨k,q(k)⟩. k∈n #+ 1}
  shows
  a: n #+ 1 → X
  {⟨k,q(k)⟩. k ∈ n}: n → X
  Init(a) = {⟨k,q(k)⟩. k ∈ n}
  a = Append({⟨k,q(k)⟩. k ∈ n},q(n))
  a = Append(Init(a), q(n))
  a = Append(Init(a), a(n))
proof -
  from assms(1) have I: n #+ 1 = succ(n) using succ_add_one(1)
    by simp
  with assms show
    a: n #+ 1 → X and {⟨k,q(k)⟩. k ∈ n}: n → X
    and II: Init(a) = {⟨k,q(k)⟩. k ∈ n}
    using set_list_append(1,2) init_def_alt by simp_all
  from assms(2,3) I have
    ∀k∈succ(n). q(k) ∈ X and a = {⟨k,q(k)⟩. k ∈ succ(n)}
    by simp_all
  then show a = Append({⟨k,q(k)⟩. k ∈ n},q(n))
    using set_list_append(3) by simp
  with II show a = Append(Init(a), q(n)) by simp
  from I have n ∈ n #+ 1 by simp
  then have {⟨k,q(k)⟩. k∈n #+ 1}(n) = q(n)
    by (rule ZF_fun_from_tot_val1)
  with assms(3) ‹a = Append(Init(a), q(n))› show a = Append(Init(a),
a(n))
    by simp
qed
```

An induction theorem for lists.

```
lemma list_induct: assumes A1: ∀b∈1→X. P(b) and
  A2: ∀b∈NELists(X). P(b) ⟶ (∀x∈X. P(Append(b,x))) and
  A3: d ∈ NELists(X)
  shows P(d)
proof -
  { fix n
    assume n∈nat
    moreover from A1 have ∀b∈succ(0)→X. P(b) by simp
    moreover have ∀k∈nat. ((∀b∈succ(k)→X. P(b)) ⟶ (∀c∈succ(succ(k))→X.
P(c)))
    proof -
      { fix k assume k ∈ nat assume ∀b∈succ(k)→X. P(b)
        have ∀c∈succ(succ(k))→X. P(c)
```

```
        proof
          fix c assume c: succ(succ(k))→X
          let b = Init(c)
          let x = c(succ(k))
          from ‹k ∈ nat› ‹c: succ(succ(k))→X› have b:succ(k)→X
            using init_props by simp
          with A2 ‹k ∈ nat› ‹∀b∈succ(k)→X. P(b)› have ∀x∈X. P(Append(b,x))
            using NELists_def by auto
          with ‹c: succ(succ(k))→X› have P(Append(b,x)) using apply_funtype
by simp
          with ‹k ∈ nat› ‹c: succ(succ(k))→X› show P(c)
            using init_props by simp
        qed
      } thus thesis by simp
    qed
    ultimately have ∀b∈succ(n)→X. P(b) by (rule ind_on_nat)
  } with A3 show thesis using NELists_def by auto
qed
```

A dual notion to `Append` is `Prepend` where we add an element to the list at the beginning of the list. We define the value of the list a prepended by an element x as x if index is 0 and $a(k-1)$ otherwise.

```
definition
  Prepend(a,x) ≡ {⟨k,if k = 0 then x else a(k #- 1)⟩. k∈domain(a) #+ 1}
```

If $a : n \to X$ is a list, then a with prepended $x \in X$ is a list as well and its first element is x.

```
lemma prepend_props:
  assumes n∈nat a:n→X x∈X
  shows Prepend(a,x):(n #+ 1)→X and Prepend(a,x)(0) = x
proof -
  let b = {⟨k,if k = 0 then x else a(k #- 1)⟩. k∈n #+ 1}
  have ∀k∈n #+ 1. (if k = 0 then x else a(k #- 1)) ∈ X
  proof -
    { fix k assume k ∈ n #+ 1
      let v = if k = 0 then x else a(k #- 1)
      { assume k≠0
        with ‹k ∈ n #+ 1› have n≠0 by auto
        from assms(1) ‹k ∈ n #+ 1› have k ∈ nat
          using elem_nat_is_nat(2) by blast
        from assms(1) have succ(n) = n #+ 1
          using succ_add_one(1) by simp
        with ‹k ∈ n #+ 1› have k∈succ(n) by simp
        with assms(1) ‹n≠0› have pred(k) ∈ n
          using pred_succ_mem by simp
        with assms(2) ‹k ∈ nat› ‹k≠0› have v∈X
          using pred_minus_one apply_funtype by simp
      }
      with assms(3) have v ∈ X by simp
```

```
    } thus thesis by simp
  qed
  then have b: (n #+ 1)→X by (rule ZF_fun_from_total)
  with assms(2) show Prepend(a,x):(n #+ 1)→X
    using func1_1_L1 unfolding Prepend_def by simp
  from assms(1) have 0 ∈ n #+ 1
    using succ_add_one(1) empty_in_every_succ by simp
  then have b(0) = (if 0 = 0 then x else a(0 #- 1))
    by (rule ZF_fun_from_tot_val1)
  with assms(2) show  Prepend(a,x)(0) = x
    using func1_1_L1 unfolding Prepend_def by simp
qed
```

When prepending an element to a list the values at positive indices do not change.

```
lemma prepend_val: assumes n∈nat a:n→X x∈X k∈n
  shows Prepend(a,x)(k #+ 1) = a(k)
proof -
  let b = {⟨k,if k = 0 then x else a(k #- 1)⟩. k∈n #+ 1}
  from assms(1,4) have k∈nat
    using elem_nat_is_nat(2) by simp
  with assms(1) have succ(n) = n #+ 1 and succ(k) = k #+ 1
    using succ_add_one(1) by auto
  with assms(1,4) have k #+ 1 ∈ n #+ 1
    using succ_ineq by simp
  from ‹k #+ 1 ∈ n #+ 1› have
    b(k #+ 1) = (if k #+ 1 = 0 then x else a((k #+ 1) #- 1))
    by (rule ZF_fun_from_tot_val1)
  with assms(2) ‹k∈nat› show thesis
    using func1_1_L1 unfolding Prepend_def by simp
qed
```

The tail of a list prepended by an element is equal to the list.

```
lemma tail_prepend: assumes n∈nat a:n→X x∈X
  shows Tail(Prepend(a,x)) = a
proof -
  let b = Prepend(a,x)
  from assms have b:(n #+ 1)→X using prepend_props(1) by simp
  with assms(1) have Tail(b):n→X using tail_props(1) by simp
  from assms ‹b:(n #+ 1)→X› have ∀k∈n. Tail(b)(k) = a(k)
    using tail_props2 prepend_val by simp
  with assms(2) ‹Tail(b):n→X› show thesis using func_eq
    by blast
qed
```

20.2 Lists and cartesian products

Lists of length n of elements of some set X can be thought of as a model of the cartesian product X^n which is more convenient in many applications.

There is a natural bijection between the space $(n+1) \rightarrow X$ of lists of length $n+1$ of elements of X and the cartesian product $(n \rightarrow X) \times X$.

```
lemma lists_cart_prod: assumes n ∈ nat
  shows {⟨x,⟨Init(x),x(n)⟩⟩. x ∈ succ(n)→X} ∈ bij(succ(n)→X,(n→X)×X)
proof -
  let f = {⟨x,⟨Init(x),x(n)⟩⟩. x ∈ succ(n)→X}
  from assms have ∀x ∈ succ(n)→X. ⟨Init(x),x(n)⟩ ∈ (n→X)×X
    using init_props succ_iff apply_funtype by simp
  then have I: f: (succ(n)→X)→((n→X)×X) by (rule ZF_fun_from_total)
  moreover from assms I have ∀x∈succ(n)→X.∀y∈succ(n)→X. f(x)=f(y)
⟶ x=y
    using ZF_fun_from_tot_val init_def finseq_restr_eq by auto
  moreover have ∀p∈(n→X)×X.∃x∈succ(n)→X. f(x) = p
  proof
    fix p assume p ∈ (n→X)×X
    let x = Append(fst(p),snd(p))
    from assms ‹p ∈ (n→X)×X› have x:succ(n)→X using append_props by
simp
    with I have f(x) = ⟨Init(x),x(n)⟩ using succ_iff ZF_fun_from_tot_val
by simp
    moreover from assms ‹p ∈ (n→X)×X› have Init(x) = fst(p) and x(n)
= snd(p)
      using init_append append_props by auto
    ultimately have f(x) = ⟨fst(p),snd(p)⟩ by auto
    with ‹p ∈ (n→X)×X› ‹x:succ(n)→X› show ∃x∈succ(n)→X. f(x) = p
by auto
  qed
  ultimately show thesis using inj_def surj_def bij_def by auto
qed
```

We can identify a set X with lists of length one of elements of X.

```
lemma singleton_list_bij: shows {⟨x,x(0)⟩. x∈1→X} ∈ bij(1→X,X)
proof -
  let f = {⟨x,x(0)⟩. x∈1→X}
  have ∀x∈1→X. x(0) ∈ X using apply_funtype by simp
  then have I: f:(1→X)→X by (rule ZF_fun_from_total)
  moreover have ∀x∈1→X.∀y∈1→X. f(x) = f(y) ⟶ x=y
  proof -
    { fix x y
      assume x:1→X y:1→X and f(x) = f(y)
      with I have x(0) = y(0) using ZF_fun_from_tot_val by auto
      moreover from ‹x:1→X› ‹y:1→X› have x = {⟨0,x(0)⟩} and y = {⟨0,y(0)⟩}

        using list_singleton_pair by auto
      ultimately have x=y by simp
    } thus thesis by auto
  qed
  moreover have ∀y∈X. ∃x∈1→X. f(x)=y
  proof
```

```
    fix y assume y∈X
    let x = {⟨0,y⟩}
    from I ‹y∈X› have x:1→X and f(x) = y
      using list_len1_singleton ZF_fun_from_tot_val pair_val by auto
    thus ∃x∈1→X. f(x)=y by auto
  qed
  ultimately show thesis using inj_def surj_def bij_def by simp
qed
```

We can identify a set of X-valued lists of length with X.

```
lemma list_singleton_bij: shows
  {⟨x,{⟨0,x⟩}⟩.x∈X} ∈ bij(X,1→X) and
  {⟨y,y(0)⟩. y∈1→X} = converse({⟨x,{⟨0,x⟩}⟩.x∈X}) and
  {⟨x,{⟨0,x⟩}⟩.x∈X} = converse({⟨y,y(0)⟩. y∈1→X})
proof -
  let f = {⟨y,y(0)⟩. y∈1→X}
  let g = {⟨x,{⟨0,x⟩}⟩.x∈X}
  have 1 = {0} by auto
  then have f ∈ bij(1→X,X) and g:X→(1→X)
    using singleton_list_bij pair_func_singleton ZF_fun_from_total
    by auto
  moreover have ∀y∈1→X.g(f(y)) = y
  proof
    fix y assume y:1→X
    have f:(1→X)→X using singleton_list_bij bij_def inj_def by simp
    with ‹1 = {0}› ‹y:1→X› ‹g:X→(1→X)› show g(f(y)) = y
      using ZF_fun_from_tot_val apply_funtype func_singleton_pair
      by simp
  qed
  ultimately show g ∈ bij(X,1→X) and f = converse(g) and g = converse(f)
    using comp_conv_id by auto
qed
```

What is the inverse image of a set by the natural bijection between X-valued singleton lists and X?

```
lemma singleton_vimage: assumes U⊆X shows {x∈1→X. x(0) ∈ U} = { {⟨0,y⟩}.
y∈U}
proof
  have 1 = {0} by auto
  { fix x assume x ∈ {x∈1→X. x(0) ∈ U}
    with ‹1 = {0}› have x = {⟨0, x(0)⟩} using func_singleton_pair by auto

  } thus {x∈1→X. x(0) ∈ U} ⊆ { {⟨0,y⟩}. y∈U} by auto
  { fix x assume x ∈ { {⟨0,y⟩}. y∈U}
    then obtain y where x = {⟨0,y⟩} and y∈U by auto
    with ‹1 = {0}› assms have x:1→X using pair_func_singleton by auto
  } thus { {⟨0,y⟩}. y∈U} ⊆ {x∈1→X. x(0) ∈ U} by auto
qed
```

A technical lemma about extending a list by values from a set.

```
lemma list_append_from: assumes A1: n ∈ nat and A2: U ⊆ n→X and A3:
V ⊆ X
  shows
  {x ∈ succ(n)→X. Init(x) ∈ U ∧ x(n) ∈ V} = (⋃y∈V.{Append(x,y).x∈U})
proof -
  { fix x assume x ∈ {x ∈ succ(n)→X. Init(x) ∈ U ∧ x(n) ∈ V}
    then have x ∈ succ(n)→X and Init(x) ∈ U and I: x(n) ∈ V
      by auto
    let y = x(n)
    from A1 and ‹x ∈ succ(n)→X›  have x = Append(Init(x),y)
      using init_props by simp
    with I and ‹Init(x) ∈ U› have x ∈ (⋃y∈V.{Append(a,y).a∈U}) by auto
  }
  moreover
  { fix x assume x ∈ (⋃y∈V.{Append(a,y).a∈U})
    then obtain a y where y∈V and a∈U and x = Append(a,y) by auto
    with A2 A3 have x: succ(n)→X using append_props by blast
    from A2 A3 ‹y∈V› ‹a∈U› have a:n→X and y∈X by auto
    with A1 ‹a∈U›  ‹y∈V› ‹x = Append(a,y)› have Init(x) ∈ U and  x(n)
∈ V
      using append_props init_append by auto
    with ‹x: succ(n)→X› have x ∈ {x ∈ succ(n)→X. Init(x) ∈ U ∧ x(n)
∈ V}
      by auto
  }
  ultimately show thesis by blast
qed

end
```

21 Formal languages

```
theory Finite_State_Machines_ZF imports FiniteSeq_ZF Finite1 ZF.CardinalArith

begin
```

21.1 Introduction

This file deals with finite state machines. The goal is to define regular languages and show that they are closed by finite union, finite intersection, complements and concatenation.

We show that the languages defined by deterministic, non-deterministic and non-deterministic with ϵ moves are equivalent.

First, a transitive closure variation on `r^* = id(field(r)) ∪ (r O r^*)`.

```
theorem rtrancl_rev:
```

```
  shows r^* = id(field(r)) ∪ (r^* O r)
proof-
  have converse(r)^* = id(field(converse(r))) ∪ (converse(r) O converse(r)^*)
    using rtrancl_unfold[of converse(r)] by auto
  then have converse(r^*) = id(field(r)) ∪ (converse(r) O converse(r^*))
    using rtrancl_converse by auto
  then have converse(r^*) = id(field(r)) ∪ (converse(r^* O r))
    using converse_comp by auto moreover
  {
    fix t assume t:t∈id(field(r)) ∪ (converse(r^* O r))
    {
      assume t∈id(field(r))
      then have t∈converse(id(field(r)) ∪ (r^* O r)) by auto
    } moreover
    {
      assume t∉id(field(r))
      with t have t∈converse(r^* O r) by auto
      then have t∈converse(id(field(r)) ∪ (r^* O r)) by auto
    }
    ultimately have t∈converse(id(field(r)) ∪ (r^* O r)) by auto
  }
  moreover
  {
    fix t assume t:t∈converse(id(field(r)) ∪ (r^* O r))
    then obtain t1 t2 where t12:t=⟨t1,t2⟩ ⟨t2,t1⟩∈id(field(r)) ∪ (r^*
O r)
      by auto
    {
      assume ⟨t2,t1⟩∈id(field(r))
      with t12(1) have t∈id(field(r)) by auto
      then have t∈id(field(r)) ∪ converse(r^* O r) by auto
    } moreover
    {
      assume ⟨t2,t1⟩∉id(field(r))
      with t12(2) have ⟨t2,t1⟩∈(r^* O r) by auto
      with t12(1) have t∈converse(r^* O r) by auto
      then have t∈id(field(r)) ∪ converse(r^* O r) by auto
    }
    ultimately have t∈id(field(r)) ∪ converse(r^* O r) by auto
  }
  ultimately have converse(id(field(r)) ∪ (r^* O r)) = converse(r^*) by
auto
  then have converse(converse(id(field(r)) ∪ (r^* O r))) = r^* using
converse_converse[OF rtrancl_type]
    by auto moreover
  {
    fix t assume t:t∈id(field(r)) ∪ (r^* O r)
    {
      assume t∈id(field(r))
```

```
      then have t:field(r)*field(r) by auto
    } moreover
    {
      assume t∉id(field(r))
      with t have t∈r^* O r by auto
      then have t:field(r)*field(r) using rtrancl_type unfolding comp_def
by auto
    } ultimately
    have t∈field(r)*field(r) by auto
  }
  ultimately show thesis using converse_converse[of id(field(r))∪(r^*
O r) field(r) λ_. field(r)] by auto
qed
```

A language is a subset of words.

```
definition
  IsALanguage (_{is a language with alphabet}_) where
  Finite(Σ) ⟹ L {is a language with alphabet} Σ ≡ L ⊆ Lists(Σ)
```

The set of all words, and the set of no words are languages.

```
lemma full_empty_language:
  assumes Finite(Σ)
  shows Lists(Σ) {is a language with alphabet} Σ
  and 0 {is a language with alphabet} Σ
  unfolding IsALanguage_def[OF assms] by auto
```

21.2 Deterministic Finite Automata

A deterministic finite state automaton is defined as a finite set of states, an initial state, a transition function from state to state based on the word and a set of final states.

```
definition
  DFSA ('(_,_,_,_'){is an DFSA for alphabet}_) where
  Finite(Σ) ⟹ (S,s₀,t,F){is an DFSA for alphabet}Σ ≡ Finite(S) ∧ s₀
∈ S ∧ F ⊆ S ∧ t:S×Σ → S
```

A finite automaton defines transitions on pairs of words and states. Two pairs are transition related if the second word is equal to the first except it is missing the last symbol, and the second state is generated by this symbol and the first state by way of the transition function.

```
definition
  DFSAExecutionRelation ({reduce D-relation}'(_,_,_'){in alphabet}_) where
  Finite(Σ) ⟹ (S,s₀,t,F){is an DFSA for alphabet}Σ ⟹
  {reduce D-relation}(S,s₀,t){in alphabet}Σ ≡ {⟨⟨w,s⟩,⟨Init(w),t⟨s,Last(w)⟩⟩⟩.
⟨w,s⟩∈NELists(Σ)×S}
```

We define a word to be fully reducible by a finite state automaton if in the transitive closure of the previous relation it is related to the pair of the empty word and a final state.

Since the empty word with the initial state need not be in `field({reduce D-relation}(S,s₀,t){in alphabet}Σ)`, we add the extra condition that $\langle\langle\emptyset, s_0\rangle, \emptyset, s_0\rangle$ is also a valid transition.

```
definition
  DFSASatisfy (_ <-D '(_,_,_,_'){in alphabet}_) where
  Finite(Σ) ⟹ (S,s0,t,F){is an DFSA for alphabet}Σ ⟹ i∈Lists(Σ) ⟹

  i <-D (S,s0,t,F){in alphabet}Σ ≡ (∃q∈F. ⟨⟨i,s0⟩,⟨0,q⟩⟩∈ ({reduce D-relation}(S,s0,t){in
alphabet}Σ)^*) ∨ (i = 0 ∧ s0∈F)
```

We define a locale for better notation

```
locale DetFinStateAuto =
  fixes S and s0 and t and F and Σ
  assumes finite_alphabet: Finite(Σ)

  assumes DFSA: (S,s0,t,F){is an DFSA for alphabet}Σ
```

We abbreviate the reduce relation to a single symbol within this locale.

```
abbreviation (in DetFinStateAuto) rD where
 rD ≡ {reduce D-relation}(S,s0,t){in alphabet}Σ
```

We abbreviate the full reduction condition to a single symbol within this locale.

```
abbreviation (in DetFinStateAuto) reduce (_{reduces}) where
  i{reduces} ≡ i <-D (S,s0,t,F){in alphabet}Σ
```

Destruction lemma about deterministic finite state automata.

```
lemma (in DetFinStateAuto) DFSA_dest:
  shows s0∈S F⊆S t:S×Σ → S Finite(S) using DFSA unfolding DFSA_def[OF
finite_alphabet] by auto
```

The set of words that reduce to final states forms a language. This is by definition.

```
lemma (in DetFinStateAuto) DFSA_language:
  shows {i∈Lists(Σ). i <-D (S,s0,t,F){in alphabet}Σ} {is a language with
alphabet}Σ
  unfolding IsALanguage_def[OF finite_alphabet] by auto
```

Define this language as an abbreviation to reduce terms

```
abbreviation (in DetFinStateAuto) LanguageDFSA
  where LanguageDFSA ≡ {i∈Lists(Σ). i <-D (S,s0,t,F){in alphabet}Σ}
```

The relation is an actual relation, but even more it is a function (hence the adjective deterministic).

```
lemma (in DetFinStateAuto) reduce_is_relation_function:
  shows relation(r_D) function(r_D) unfolding DFSAExecutionRelation_def[OF
finite_alphabet DFSA]
   relation_def function_def by auto
```

The relation, that is actually a function has the following domain and range:

```
lemma (in DetFinStateAuto) reduce_function:
shows r_D:NELists(Σ)×S→Lists(Σ)×S
proof-
  from DFSA have T:t:S×Σ → S unfolding DFSA_def[OF finite_alphabet]
by auto
  {
    fix x assume x∈r_D
    then obtain l s where x:l ∈ NELists(Σ) s ∈ S x = ⟨⟨l,s⟩,⟨Init(l),t⟨s,Last(l)⟩⟩⟩
unfolding
      DFSAExecutionRelation_def[OF finite_alphabet DFSA] by auto
    from x(1) have Init(l) ∈ Lists(Σ) using init_NElist(1) by auto more-
over
    from x(1) have Last(l) ∈ Σ using last_type by auto
    with x(2) have t⟨s,Last(l)⟩ ∈ S using apply_type[OF T] by auto
    moreover note x
    ultimately have x∈(NELists(Σ)×S)×(Lists(Σ)×S) by auto
  }
  then have r:r_D∈Pow((NELists(Σ)×S)×(Lists(Σ)×S)) by auto moreover
  {
    fix x assume x∈NELists(Σ)×S
    then obtain l s where x:l∈NELists(Σ) s∈S x=⟨l,s⟩ by auto
    then have ⟨⟨l,s⟩,⟨Init(l),t⟨s,Last(l)⟩⟩⟩∈r_D unfolding
      DFSAExecutionRelation_def[OF finite_alphabet DFSA] by auto
    with x(3) have x∈domain(r_D) unfolding domain_def by auto
  }
  then have NELists(Σ)×S ⊆ domain(r_D) by auto moreover
  note reduce_is_relation_function(2)
  ultimately show thesis unfolding Pi_def by auto
qed
```

The field of the relation contains all pairs with non-empty words, but we cannot assume that it contains all pairs.

```
corollary (in DetFinStateAuto) reduce_field:
shows field(r_D) ⊆ Lists(Σ)×S NELists(Σ)×S ⊆ field(r_D)
proof-
  from DFSA have T:t:S×Σ → S unfolding DFSA_def[OF finite_alphabet]
by auto
  {
    fix x assume x∈field(r_D)
    then have E:∃y. ⟨x,y⟩∈r_D ∨ ⟨y,x⟩∈r_D unfolding domain_def range_def
field_def by auto
    {
      assume ∃y. ⟨x,y⟩∈r_D
```

```
      then have x∈NELists(Σ)×S unfolding DFSAExecutionRelation_def[OF
finite_alphabet DFSA]
        by auto
      then have x∈Lists(Σ)×S unfolding Lists_def NELists_def by auto
    } moreover
    {
      assume ¬(∃y. ⟨x,y⟩∈r_D)
      with E have ∃y.⟨y,x⟩∈r_D by auto
      then obtain u v where y:u∈NELists(Σ) v∈S x=⟨Init(u),t⟨v,Last(u)⟩⟩
        unfolding DFSAExecutionRelation_def[OF finite_alphabet DFSA] by
auto
      from y(1) have Init(u) ∈ Lists(Σ) using init_NElist(1) by auto
moreover
      from y(1) have Last(u) ∈ Σ using last_type by auto
      with y(2) have t⟨v,Last(u)⟩ ∈ S using apply_type[OF T] by auto
      moreover note y(3) ultimately have x∈Lists(Σ)×S by auto
    }
    ultimately have x∈Lists(Σ)×S by auto
  }
  then show field(r_D) ⊆ Lists(Σ)×S by auto
  show NELists(Σ)×S ⊆ field(r_D)
    using domain_of_fun[OF reduce_function] unfolding field_def by auto
qed
```

If a word is a reduced version of an other, then it can be encoded as a restriction.

```
lemma (in DetFinStateAuto) seq_is_restriction:
  fixes w s u v
  assumes ⟨⟨w,s⟩,⟨u,v⟩⟩∈r_D^*
  shows restrict(w,domain(u)) = u
proof-
  from assms have ⟨w,s⟩∈field(r_D) using rtrancl_field[of r_D] relation_field_times_field[OF
relation_rtrancl[of r_D]] by auto
  then have w∈Lists(Σ) using reduce_field(1) by auto
  then obtain n where w:n→Σ unfolding Lists_def by auto
  then have restrict(w,n) = w w:n→Σ using restrict_idem unfolding Pi_def
by auto
  then have base:restrict(w,domain(w)) = w using domain_of_fun by auto
  {
    fix y z
    assume as:⟨⟨w, s⟩, y⟩ ∈ r_D^* ⟨y, z⟩ ∈ r_D restrict(w, domain(fst(y)))
= fst(y)
    from as(1) have y:field(r_D)  using rtrancl_field[of r_D] relation_field_times_field[OF
relation_rtrancl[of r_D]] by auto
    then obtain y1 y2 where y:y=⟨y1,y2⟩ y1∈Lists(Σ) y2∈S using reduce_field(1)
      by auto
    with as(2) have z:z=⟨Init(y1),t⟨y2,Last(y1)⟩⟩ y1∈NELists(Σ) unfold-
ing DFSAExecutionRelation_def[OF finite_alphabet DFSA]
      by auto
```

```
    then have fst(z) = Init(y1) by auto
    with z(2) have S:succ(domain(fst(z))) = domain(y1) using init_NElist(2)
by auto
    from as(3) y(1) have restrict(w,domain(y1)) = y1 by auto
    then have restrict(restrict(w,domain(y1)),pred(domain(y1))) = Init(y1)
unfolding Init_def by auto
    then have w:restrict(w,domain(y1)∩pred(domain(y1))) = Init(y1) us-
ing restrict_restrict by auto
    from z(2) obtain q where q:domain(y1) = succ(q) q∈nat using domain_of_fun
unfolding NELists_def by auto
    then have pred(domain(y1)) ⊆ domain(y1) using pred_succ_eq by auto
    then have domain(y1) ∩ pred(domain(y1)) = pred(domain(y1)) by auto
    with w have restrict(w,pred(domain(y1))) = Init(y1) by auto more-
over
    from q z(2) init_props(1)[of _ y1 Σ] have domain(Init(y1)) = pred(domain(y1))

      using domain_of_fun[of y1 _ λ_. Σ] domain_of_fun[of Init(y1) _ λ_.
Σ]
      unfolding NELists_def by auto ultimately
    have restrict(w,domain(Init(y1))) = Init(y1) by auto
    with z(1) have restrict(w,domain(fst(z))) = fst(z) by auto
  }
  then have reg:∀y z. ⟨⟨w, s⟩, y⟩ ∈ r_D^* ⟶
            (⟨y, z⟩ ∈ r_D ⟶
            (restrict(w, domain(fst(y))) = fst(y) ⟶
            restrict(w, domain(fst(z))) = fst(z))) by auto
  have restrict(w, domain(fst(⟨u, v⟩))) = fst(⟨u, v⟩)
  proof(rule rtrancl_induct[of ⟨w,s⟩ ⟨u,v⟩ r_D λq. restrict(w,domain(fst(q)))
= fst(q)])
    from base show restrict(w,domain(fst(⟨w,s⟩))) = fst(⟨w,s⟩) by auto
    from assms show ⟨⟨w, s⟩, u, v⟩ ∈ r_D^* by auto
    {
      fix y z
      assume as:⟨⟨w, s⟩, y⟩ ∈ r_D^* ⟨y, z⟩ ∈ r_D restrict(w, domain(fst(y)))
= fst(y)
      from as(1) have y:field(r_D)  using rtrancl_field[of r_D] relation_field_times_field[O
relation_rtrancl[of r_D]] by auto
      then obtain y1 y2 where y:y=⟨y1,y2⟩ y1∈Lists(Σ) y2∈S using reduce_field(1)
        by auto
      with as(2) have z:z=⟨Init(y1),t⟨y2,Last(y1)⟩⟩ y1∈NELists(Σ) un-
folding DFSAExecutionRelation_def[OF finite_alphabet DFSA]
        by auto
      then have fst(z) = Init(y1) by auto
      with z(2) have S:succ(domain(fst(z))) = domain(y1) using init_NElist(2)
by auto
      from as(3) y(1) have restrict(w,domain(y1)) = y1 by auto
      then have restrict(restrict(w,domain(y1)),pred(domain(y1))) = Init(y1)
unfolding Init_def by auto
      then have w:restrict(w,domain(y1)∩pred(domain(y1))) = Init(y1)
```

```
using restrict_restrict by auto
      from z(2) obtain q where q:domain(y1) = succ(q) q∈nat using domain_of_fun
unfolding NELists_def by auto
      then have pred(domain(y1)) ⊆ domain(y1) using pred_succ_eq by auto
      then have domain(y1) ∩ pred(domain(y1)) = pred(domain(y1)) by auto
      with w have restrict(w,pred(domain(y1))) = Init(y1) by auto more-
over
      from q z(2) init_props(1)[of _ y1 Σ] have domain(Init(y1)) = pred(domain(y1))

        using domain_of_fun[of y1 _ λ_. Σ] domain_of_fun[of Init(y1) _
λ_. Σ]
        unfolding NELists_def by auto ultimately
      have restrict(w,domain(Init(y1))) = Init(y1) by auto
      with z(1) show restrict(w,domain(fst(z))) = fst(z) by auto
    }
  qed
  then show thesis by auto
qed

lemma (in DetFinStateAuto) relation_deteministic:
  assumes ⟨⟨w,s⟩,⟨u,v⟩⟩∈r_D^* ⟨⟨w,s⟩,⟨u,m⟩⟩∈r_D^*
  shows v=m
proof-
  let P=λy. ∀q1 q2. ⟨⟨w,s⟩,⟨q1,q2⟩⟩∈r_D^* ⟶ fst(y) = q1 ⟶ snd(y) = q2
  {
    fix q1 q2 assume ⟨⟨w, s⟩, q1, q2⟩ ∈ r_D^* fst(⟨w, ss⟩) = q1
    then have ⟨⟨w, s⟩, w, q2⟩ ∈ r_D^* by auto
    then have ⟨⟨w, s⟩, w, q2⟩ ∈ id(field(r_D)) ∪ (r_D^* O r_D) using rtrancl_rev
by auto
    then have A:s=q2 ∨ ⟨⟨w, s⟩, w, q2⟩:(r_D^* O r_D) by auto
    {
      assume s≠q2
      with A have ⟨⟨w, s⟩, w, q2⟩:(r_D^* O r_D) by auto
      then obtain b where b:⟨⟨w,s⟩,b⟩:r_D ⟨b,w,q2⟩:r_D^* unfolding compE
by auto
      from b(1) have b=⟨Init(w),t⟨s,Last(w)⟩⟩ and w:w:NELists(Σ) un-
folding DFSAExecutionRelation_def[OF finite_alphabet DFSA] by auto
      with b(2) have restrict(Init(w),domain(w)) = w using seq_is_restriction
by auto
      then have domain(Init(w))∩domain(w) = domain(w) using domain_restrict[of
Init(w) domain(w)] by auto
      with w have e:domain(Init(w))∩domain(w) = succ(domain(Init(w)))
using init_NElist(2)[of w] by auto
      {
        fix tt assume t:tt:succ(domain(Init(w)))
        with e have tt:domain(Init(w))∩domain(w) by auto
        then have tt:domain(Init(w)) by auto
      }
      then have succ(domain(Init(w)))⊆domain(Init(w)) by auto
```

```
      then have domain(Init(w)) ∈ domain(Init(w)) by auto
      then have False using mem_irrefl[of domain(Init(w))] by auto
    }
    then have s=q2 by auto
    then have snd(⟨w,s⟩) = q2 by auto
  }
  then have P0:P(⟨w,s⟩) by auto
  {
    fix y z assume y:⟨⟨w,s⟩,y⟩:r_D^* ⟨y,z⟩:r_D P(y)
    {
      fix q1 q2 assume z:⟨⟨w, s⟩, q1, q2⟩ ∈ r_D^* fst(z) = q1
      from this(1) have ⟨⟨w, s⟩, q1, q2⟩ ∈ id(field(r_D)) ∪ (r_D O r_D^*)
using rtrancl_unfold by auto
      then have A:(s=q2 ∧ w=q1) ∨ ⟨⟨w, s⟩, q1, q2⟩:(r_D O r_D^*) by auto
      from y(2) obtain y1 y2 where zz:y=⟨y1,y2⟩ y1:NELists(Σ) y2∈S z=⟨Init(y1),t⟨y2,Last(y
        unfolding DFSAExecutionRelation_def[OF finite_alphabet DFSA] by
auto
      {
        assume w:w=q1
        with z(2) zz(4) have w=Init(y1) by auto
        with y(1) zz(1) have restrict(Init(y1),domain(y1)) = y1 using
seq_is_restriction
          by auto
        then have domain(Init(y1))∩domain(y1) = domain(y1) using domain_restrict[of
Init(y1) domain(y1)] by auto
        with zz(2) have e:domain(Init(y1))∩domain(y1) = succ(domain(Init(y1)))
using init_NElist(2)[of y1] by auto
        {
          fix tt assume t:tt:succ(domain(Init(y1)))
          with e have tt:domain(Init(y1))∩domain(y1) by auto
          then have tt:domain(Init(y1)) by auto
        }
        then have succ(domain(Init(y1)))⊆ domain(Init(y1)) by auto
        then have domain(Init(y1)) ∈ domain(Init(y1)) by auto
        then have False using mem_irrefl[of domain(Init(y1))] by auto
      }
      with A have ⟨⟨w, s⟩, q1, q2⟩:(r_D O r_D^*) by auto
      then obtain pp where pp:⟨⟨w, s⟩, pp⟩:r_D^* ⟨pp, q1, q2⟩:r_D unfold-
ing compE by auto
      from this(2) obtain ppL ppS where ppl:ppL:NELists(Σ) ppS∈S pp=⟨ppL,ppS⟩
        q1=Init(ppL) q2=t⟨ppS,Last(ppL)⟩ unfolding DFSAExecutionRelation_def[OF
finite_alphabet DFSA] by auto
      from this(3) pp(1) have rr:restrict(w,domain(ppL)) = ppL using
seq_is_restriction by auto
      then have r:restrict(w,domain(ppL))pred(domain(ppL)) = Last(ppL)
unfolding Last_def by auto
      from ppl(1) obtain q where q:ppL:succ(q) → Σ q∈nat unfolding
NELists_def by blast
      from q(1) have D:domain(ppL) = succ(q) using func1_1_L1[of ppL]
```

```
by auto moreover
      from D have pred(domain(ppL)) = q using pred_succ_eq by auto
      then have pred(domain(ppL)) ∈ succ(q) by auto
      ultimately have pred(domain(ppL)) ∈ domain(ppL) by auto
      with restrict r have W:wpred(domain(ppL)) = Last(ppL) by auto
      from y(1) zz(1) have restrict(w,domain(y1)) = y1 using seq_is_restriction
by auto
      moreover note rr moreover
      from q have Init(ppL):q → Σ using init_props(1) by auto
      then have DInit:domain(Init(ppL)) = q using func1_1_L1 by auto
      from zz(4) z(2) have q1=Init(y1) by auto
      with ppl(4) have Init(ppL) = Init(y1) by auto
      with DInit have Dy1:domain(Init(y1)) = q by auto
      from zz(2) obtain o where o:o∈nat y1:succ(o) → Σ unfolding NELists_def
by auto
      then have Init(y1):o→Σ using init_props(1) by auto
      with Dy1 have q=o using func1_1_L1 by auto
      with o(2) have y1:succ(q)→Σ by auto moreover
      note q(1) ultimately have y1=ppL using func1_1_L1 by auto more-
over
      then have fst(y) = ppL using zz(1) by auto
      with y(3) pp(1) ppl(3) have snd(y) = ppS by auto
      with zz(1) have y2= ppS by auto moreover
      note zz(4) ultimately have z=⟨Init(ppL),t⟨ppS,Last(ppL)⟩⟩ by auto
      then have snd(z) = t⟨ppS,Last(ppL)⟩ by auto
      with ppl(5) have snd(z) = q2 by auto
    }
    then have P(z) by auto
  }
  then have R:⋀y z. ⟨⟨w,s⟩,y⟩∈r_D^* ⟹ ⟨y,z⟩∈r_D ⟹ P(y) ⟹ P(z) by
auto
  then have P(⟨u,v⟩) using rtrancl_induct[of ⟨w,s⟩ ⟨u,v⟩ r_D λy. ∀q1 q2.
⟨⟨w,s⟩,⟨q1,q2⟩⟩:r_D^* ⟶ fst(y) = q1 ⟶ snd(y) = q2]
    P0 assms(1) by auto
  then show thesis using assms(2) by auto
qed
```

Any non-empty word can be reduced to the empty string, but it does not always end in a final state.

```
lemma (in DetFinStateAuto) endpoint_exists:
  assumes w∈NELists(Σ)
  shows ∃q ∈ S. ⟨⟨w,s_0⟩,⟨0,q⟩⟩ ∈ r_D^*
proof-
  let P=λk. ∀y∈Lists(Σ). domain(y) = k ⟶ y=0 ∨ (∀ss∈S. (∃q∈S. ⟨⟨y,ss⟩,⟨0,q⟩⟩∈r_D^*))
  {
    fix y assume y∈Lists(Σ) domain(y) = 0
    with assms have y=0 unfolding Lists_def using domain_of_fun by auto
    then have y=0 ∨ (∀ss∈S. (∃q∈S. ⟨⟨y,ss⟩,⟨0,q⟩⟩∈r_D^*)) by auto
  }
```

```
  then have base:P(0) by auto
  {
    fix k assume hyp:P(k) k∈nat
    {
      fix y assume as:y∈Lists(Σ) domain(y) = succ(k)
      from as have y:succ(k)→Σ unfolding Lists_def using domain_of_fun
by auto
      with hyp(2) have y:y:NELists(Σ) unfolding NELists_def by auto
      then have Init(y):Lists(Σ) succ(domain(Init(y))) = domain(y) us-
ing init_NElist by auto
      with as(2) have D:Init(y):Lists(Σ) domain(Init(y)) = k using pred_succ_eq
by auto
      with hyp(1) have d:Init(y) = 0 ∨ (∀ss∈S. (∃q∈S. ⟨⟨Init(y),ss⟩,⟨0,q⟩⟩∈r_D^*))
by auto
      {
        assume iy0:Init(y) = 0
        {
          fix ss assume ss∈S
          with iy0 y have ⟨⟨y,ss⟩,0,t⟨ss,Last(y)⟩⟩∈r_D unfolding DFSAExecutionRelation_def[OF
finite_alphabet DFSA]
            by auto
          then have ⟨⟨y,ss⟩,0,t⟨ss,Last(y)⟩⟩∈r_D^* using r_into_rtrancl
by auto
          moreover from `ss∈S` have t⟨ss,Last(y)⟩ ∈S using apply_type[OF
DFSA_dest(3)]
            last_type[OF y] by auto
          ultimately have ∃q∈S. ⟨⟨y,ss⟩,0,q⟩:r_D^* by auto
        }
        then have ∀ss∈S. ∃q∈S. ⟨⟨y,ss⟩,0,q⟩:r_D^* by auto
        then have y = 0 ∨ (∀ss∈S. ∃q∈S. ⟨⟨y,ss⟩,0,q⟩:r_D^*) by auto
      } moreover
      {
        assume qS:∀ss∈S. ∃q∈S. ⟨⟨Init(y),ss⟩,⟨0,q⟩⟩∈r_D^*
        {
          fix ss assume ss∈S
          with y have ⟨⟨y,ss⟩,Init(y),t⟨ss,Last(y)⟩⟩∈r_D unfolding DFSAExecutionRelation_def[
finite_alphabet DFSA]
            by auto
          moreover from `ss∈S` y have t⟨ss,Last(y)⟩ ∈S using apply_type[OF
DFSA_dest(3)]
            last_type by auto
          with qS have ∃q∈S. ⟨⟨Init(y),t⟨ss,Last(y)⟩⟩,⟨0,q⟩⟩∈r_D^* by auto
          then obtain q where q∈S ⟨⟨Init(y),t⟨ss,Last(y)⟩⟩,⟨0,q⟩⟩∈r_D^*
by auto
          ultimately have ⟨⟨y,ss⟩,⟨0,q⟩⟩∈r_D^* q∈S using rtrancl_into_trancl2
            trancl_into_rtrancl by auto
          then have ∃q∈S. ⟨⟨y,ss⟩,⟨0,q⟩⟩∈r_D^* by auto
        }
        then have y = 0 ∨ (∀ss∈S. ∃q∈S. ⟨⟨y,ss⟩,0,q⟩:r_D^*)  by auto
```

```
      } ultimately
      have y = 0 ∨ (∀ss∈S. ∃q∈S. ⟨⟨y,ss⟩,0,q⟩:r_D^*) using d by auto
    }
    then have P(succ(k)) by auto
  }
  then have ind:⋀k. k∈nat ⟹ P(k) ⟹ P(succ(k)) by blast
  have dom:domain(w) ∈ nat using assms unfolding NELists_def using domain_of_fun
by auto
  from ind have P(domain(w)) using nat_induct[of _ P, OF dom base] by
auto
  with assms have (∀ss∈S. ∃q∈S. ⟨⟨w, ss⟩, 0, q⟩ ∈ r_D^*)
    using non_zero_List_func_is_NEList by auto
  then show thesis using DFSA_dest(1) by auto
qed
```

Example of Finite Automaton of binary lists starting with 0 and ending with 1

```
locale ListFrom0To1
begin
```

Empty state

```
definition empty where
  empty ≡ 2
```

The string starts with 0 state

```
definition ends0 where
  ends0 ≡ succ(2)
```

The string ends with 1 state

```
definition starts1 where
  starts1 ≡ 1
```

The string ends with 0 state

```
definition starts0 where
  starts0 ≡ 0
```

The states are the previous 4 states. They are encoded as natural numbers to make it easier to reason about them, and as human readable variable names to make it easier to understand.

```
definition states where
  states ≡ {empty, starts0, starts1, ends0}
```

The final state is `starts0`

```
definition finalStates where
finalStates ≡ {starts0}
```

The transition function is defined as follows:

From the empty state, we transition to state starts1 in case there is a 1 and to state ends0 in case there is a 0.

From the state ends0 we stay in it.

From the states starts1 and starts0 we transition to starts0 in case there is a 0, and to starts1 in case there is a 1.

```
definition transFun where
transFun ≡ {⟨⟨empty,1⟩,starts1⟩,⟨⟨empty,0⟩,ends0⟩}∪
               {⟨⟨ends0,x⟩,ends0⟩. x∈2}∪
               {⟨⟨starts1,0⟩,starts0⟩,⟨⟨starts1,1⟩,starts1⟩,
                ⟨⟨starts0,0⟩,starts0⟩,⟨⟨starts0,1⟩,starts1⟩}
```

Add lemmas to simplify

```
lemmas from0To1[simp] = states_def empty_def transFun_def finalStates_def
ends0_def starts1_def starts0_def
```

Interpret the example as a deterministic finite state automaton

```
interpretation dfsaFrom0To1: DetFinStateAuto states empty transFun finalStates
2 unfolding DetFinStateAuto_def apply safe
  using Finite_0 apply simp
proof-
  have finA:Finite(2) using nat_into_Finite[of 2] by auto
  have finS:Finite(states) using nat_into_Finite by auto
  moreover have funT:transFun:states×2 → states unfolding Pi_def function_def
by auto
  moreover have finalStates ⊆ states by auto
  moreover have empty∈ states by auto
  ultimately show (states,empty,transFun,finalStates){is an DFSA for alphabet}2

    unfolding DFSA_def[OF finA] by auto
qed
```

Abbreviate the relation to something readable.

```
abbreviation r0to1 (r{0.*1}) where
  r{0.*1} ≡ dfsaFrom0To1.r_D
```

If a word reaches the state starts0, it does not move from it.

```
lemma invariant_state_3:
  fixes w u y
  assumes ⟨⟨w,ends0⟩,⟨u,y⟩⟩∈r{0.*1}^*
  shows y = ends0
proof-
  have finA:Finite(2) by auto
  have funT:transFun:states×2→ states
    using dfsaFrom0To1.DFSA_dest(3) .
  have snd(⟨u,y⟩) = ends0
  proof(rule rtrancl_induct[of ⟨w,ends0⟩
        ⟨u,y⟩ r{0.*1} λt. snd(t) = ends0])
```

```
    show snd(⟨w, ends0⟩) = ends0 by auto
    from assms show ⟨⟨w, ends0⟩, u, y⟩ ∈ r{0.*1}^* .
    {
      fix y z assume as:⟨⟨w, ends0⟩, y⟩ ∈ r{0.*1}^* ⟨y, z⟩ ∈ r{0.*1} snd(y)
= ends0
      from as(3,2) obtain y1 where yy:y=⟨y1,ends0⟩ y1∈NELists(2)
        z=⟨Init(y1),transFun⟨ends0,Last(y1)⟩⟩
        unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
        by auto
      from yy(2) have Last(y1)∈2 using last_type by auto
      then have ⟨⟨ends0,Last(y1)⟩,ends0⟩∈transFun by auto
      then have transFun⟨ends0,Last(y1)⟩ = ends0
        using apply_equality[OF _ funT, of _ ends0] by auto
      with yy(3) have z=⟨Init(y1),ends0⟩ by auto
      then show snd(z) = ends0 by auto
    }
  qed
  then show thesis by auto
qed
```

If the string starts in 0 and has reached states `starts0` or `starts1`; then it reduces to `starts0`.

```
lemma invariant_state_0_1:
  fixes w
  assumes w∈NELists(2) w0 = 0
  shows ⟨⟨w,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ⟨⟨w,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^*
proof-
  from assms(1) obtain n where w:n:nat w:succ(n)→2 unfolding NELists_def
by auto
  then have dom:domain(w)∈nat using domain_of_fun by auto
  {
    fix q assume q∈NELists(2)
    then obtain m where m∈nat q:succ(m)→2 unfolding NELists_def by auto
    then have domain(q)≠0 using domain_of_fun by auto
  }
  then have base:∀w. w ∈ NELists(2) ∧ w0 = 0 ∧ domain(w) = 0 ⟶
        ⟨⟨w, starts0⟩, 0, starts0⟩ ∈ r{0.*1}^* ∧ ⟨⟨w, starts1⟩, 0, starts0⟩
∈ r{0.*1}^* by auto
  from w(2) have domN0:domain(w)≠ 0 using domain_of_fun by auto
  have finA:Finite(2) using nat_into_Finite[of 2] by auto
  have funT:transFun:states×2→states unfolding Pi_def function_def by
auto
  have t00:transFun⟨starts0,0⟩ = starts0 using funT apply_equality[of
⟨starts0,0⟩ starts0 transFun states×2 λ_. states] by auto
  have t01:transFun⟨starts1,0⟩ = starts0 using funT apply_equality[of
⟨starts1,0⟩ starts0 transFun states×2 λ_. states] by auto
  have t10:transFun⟨starts0,1⟩ = starts1 using funT apply_equality[of
⟨starts0,1⟩ starts1 transFun states×2 λ_. states] by auto
  have t11:transFun⟨starts1,1⟩ = starts1 using funT apply_equality[of
```

```
⟨starts1,1⟩ starts1 transFun states×2 λ_. states] by auto
  {
    fix ka assume kaNat:ka:nat
    assume k:∀w. w ∈ NELists(2) ∧ w0 = 0 ∧ domain(w) = ka ⟶ (⟨⟨w,starts0⟩,⟨0,starts0⟩⟩∈r
∧ ⟨⟨w,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^*)
    {
      fix y assume y:y∈NELists(2) y0 = 0 domain(y) = succ(ka)
      from y(1) obtain s where s:y:succ(s)→2 s∈nat unfolding NELists_def
by auto
      then have L:Last(y) = ys using last_seq_elem by auto
      then have last_2:Last(y) ∈ 2 using apply_type[OF s(1), of s] by
auto
      {
        assume ka:ka =0
        with y(3) have pred(domain(y)) = 0 using pred_succ_eq by auto
        then have y00:y0 = 0 using y(2) unfolding Last_def by auto
        then have ⟨⟨y,starts0⟩,⟨Init(y),transFun⟨starts0,Last(y)⟩⟩⟩∈r{0.*1}
unfolding
          DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
          using y(1) by auto
        with last_2 t00 t10 have ⟨⟨y,starts0⟩,⟨Init(y),Last(y)⟩⟩∈r{0.*1}
by auto moreover
        from ka y(3) s(1) have Init(y):0→2 using init_props(1)[OF s(2,1)]
          domain_of_fun[OF s(1)] by auto
        then have y0:Init(y) = 0 unfolding Pi_def by auto moreover
        from ka s(1) y(3) have s=0 using domain_of_fun by auto
        with L have LL:y0 = Last(y) by auto moreover note y(2)
        ultimately have ⟨⟨y,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1} by auto
        then have A:⟨⟨y,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* using r_into_rtrancl
by auto
        note t01 moreover have ⟨⟨y,starts1⟩,⟨Init(y),transFun⟨starts1,Last(y)⟩⟩⟩
∈ r{0.*1}
          unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
using y(1,2) by auto
        moreover note t11 last_2 y0 LL y(2) ultimately have ⟨⟨y,starts1⟩,⟨0,starts0⟩⟩
∈ r{0.*1} by auto
        then have B:⟨⟨y,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^* using r_into_rtrancl
by auto
        with A have ⟨⟨y,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ∧ ⟨⟨y,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^*
by auto
      } moreover
      {
        assume ka:ka≠0
        with kaNat obtain u where u:ka=succ(u) u∈nat using Nat_ZF_1_L3
by auto
        from y(3) s(1) kaNat s(2) have s=ka using domain_of_fun succ_inject
by auto
        with u(1) have s=succ(u) by auto moreover
        have Init(y):s→2 using init_props(1)[OF s(2,1)].
```

```
        ultimately have yu:Init(y):succ(u)→2 by auto
        with u have Init(y)∈NELists(2) domain(Init(y)) = ka
          using domain_of_fun unfolding NELists_def by auto
        moreover from yu have Init(y)0 = 0 using init_props(2)[OF nat_succI[OF
u(2)], of y 2]
          s(1) `s=succ(u)` empty_in_every_succ[OF u(2)] y(2) by auto
        moreover note k ultimately have ⟨⟨Init(y),starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^*

          ⟨⟨Init(y),starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^* by auto
        then have A:∀x∈{starts0, starts1}. ⟨⟨Init(y),x⟩,⟨0,starts0⟩⟩∈r{0.*1}^*
by auto
        have Q:⟨⟨y,starts0⟩,⟨Init(y),transFun⟨starts0,Last(y)⟩⟩⟩∈r{0.*1}
using y(2)
          unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
using y(1) by auto
        {
          assume as:Last(y) = 0
          with Q t00 have ⟨⟨y,starts0⟩,⟨Init(y),starts0⟩⟩∈r{0.*1} by auto
        } moreover
        {
          assume as:Last(y) = 1
          with Q t10 have ⟨⟨y,starts0⟩,⟨Init(y),starts1⟩⟩∈r{0.*1} by auto
        }
        moreover note last_2 ultimately have
        (⟨⟨y,starts0⟩,⟨Init(y),starts0⟩⟩∈r{0.*1}) ∨ (⟨⟨y,starts0⟩,⟨Init(y),starts1⟩⟩∈r{0.*1})
by auto
        with A have B:⟨⟨y,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* using rtrancl_into_trancl2
trancl_into_rtrancl by auto
        have Q:⟨⟨y,starts1⟩,⟨Init(y),transFun⟨starts1,Last(y)⟩⟩⟩∈r{0.*1}
using y(2)
          unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
using y(1) by auto
        {
          assume as:Last(y) = 0
          with Q t01 have ⟨⟨y,starts1⟩,⟨Init(y),starts0⟩⟩∈r{0.*1} by auto
        } moreover
        {
          assume as:Last(y) = 1
          with Q t11 have ⟨⟨y,starts1⟩,⟨Init(y),starts1⟩⟩∈r{0.*1} by auto
        }
        moreover note last_2 ultimately have
        (⟨⟨y,starts1⟩,⟨Init(y),starts0⟩⟩∈r{0.*1}) ∨ (⟨⟨y,starts1⟩,⟨Init(y),starts1⟩⟩∈r{0.*1})
by auto
        with A have ⟨⟨y,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^* using rtrancl_into_trancl2
trancl_into_rtrancl by auto
        with B have rr:⟨⟨y,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ∧ ⟨⟨y,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1
by auto
      }
      ultimately have ⟨⟨y,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ∧ ⟨⟨y,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}
```

```
by auto
    }
    then have ∀w. w ∈ NELists(2) ∧ w0 = 0 ∧ domain(w) = succ(ka) ⟶
(⟨⟨w,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ∧ ⟨⟨w,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^*)
by auto
  }
  then have rule:∀k∈nat.
       (∀w. w ∈ NELists(2) ∧ w0 = 0 ∧ domain(w) = k ⟶
            ⟨⟨w, starts0⟩, 0, starts0⟩ ∈ r{0.*1}^* ∧ ⟨⟨w, starts1⟩, 0,
starts0⟩ ∈ r{0.*1}^*) ⟶
       (∀w. w ∈ NELists(2) ∧ w0 = 0 ∧ domain(w) = succ(k) ⟶
            ⟨⟨w, starts0⟩, 0, starts0⟩ ∈ r{0.*1}^* ∧ ⟨⟨w, starts1⟩, 0,
starts0⟩ ∈ r{0.*1}^*) by blast
  from ind_on_nat[of domain(w) λt . ∀w. w∈NELists(2) ∧ w0 =0 ∧ domain(w)
= t ⟶ (⟨⟨w,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ∧ ⟨⟨w,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^*),
OF dom base rule]
  show R:⟨⟨w,starts0⟩,⟨0,starts0⟩⟩∈r{0.*1}^* ⟨⟨w,starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^*
using assms(2,1) by auto
qed
```

A more readable reduction statement

```
abbreviation red (_{reduces in 0.*1}) where
  i{reduces in 0.*1} ≡ dfsaFrom0To1.reduce(i)
```

Any list starting with 0 and ending in 1 reduces.

```
theorem starts1ends0_DFSA_reduce:
  fixes i
  assumes i∈Lists(2) and i0=0 and Last(i) = 1
  shows i{reduces in 0.*1}
proof-
  from assms(1) obtain tt where t:tt∈nat i:tt→2 unfolding Lists_def
by auto
  then have domNat:domain(i) = tt using domain_of_fun by auto
  {
    assume domain(i) = 0 moreover
    from assms(3) have ⋃(i{Arith.pred(domain(i))})=1 unfolding Last_def
apply_def by auto
    then have i≠0 by auto
    ultimately have False using t domain_of_fun by auto
  }
  with domNat t(1) obtain y where y:domain(i) = succ(y) y∈nat using Nat_ZF_1_L3
by auto
  with domNat t(2) have iList:i∈NELists(2) unfolding NELists_def by auto
  have finA:Finite(2) using nat_into_Finite[of 2] by auto
  have funT:transFun:states×2→states unfolding Pi_def function_def by
auto
  have ⟨⟨i,empty⟩,⟨Init(i),transFun⟨empty,Last(i)⟩⟩⟩:r{0.*1} using iList
unfolding
      DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA] by auto
```

```
  moreover have transFun⟨empty,Last(i)⟩ = 1 using apply_equality[OF _
funT] assms(3)
    by auto
  ultimately have U:⟨⟨i,empty⟩,⟨Init(i),starts1⟩⟩:r{0.*1} by auto
  {
    assume y = 0
    with y(1) t(2) have iFun:i:1→2 using domain_of_fun by auto
    then have i = {⟨0,i0⟩} using fun_is_set_of_pairs[of i 1 2] by auto
    with assms(2) have ii:i={⟨0,0⟩} by auto
    then have ∀y. ∃x∈{Arith.pred(domain(i))}. ⟨x, y⟩ ∈ i ⟶ y=0 by
auto
    moreover from assms(3) have eq:1 = ipred(domain(i)) unfolding Last_def
by auto
    from iFun have domain(i) = 1 using domain_of_fun by auto
    then have pred(domain(i)) = 0 using pred_succ_eq by auto
    then have ipred(domain(i)) = i0 by auto
    with eq have 1 = i0 by auto
    then have False using assms(2) by auto
  }
  then have y≠0 by auto
  then obtain k where yy:y= succ(k) k∈nat using Nat_ZF_1_L3 y(2) by
auto
  with iList y(1) have iss:i:succ(succ(k)) → 2 unfolding NELists_def
using domain_of_fun by auto
  then have Init(i):succ(k) → 2 using init_props(1) nat_succI[OF yy(2)]
by auto
  then have Init(i)∈NELists(2) unfolding NELists_def using yy(2) by auto
moreover
  have 0∈succ(k) using empty_in_every_succ yy(2) by auto
  with iss assms(2) have Init(i)0 = 0 using init_props(2)[of succ(k)
i 2] nat_succI[OF yy(2)]
    by auto
  ultimately have ⟨⟨Init(i),starts1⟩,⟨0,starts0⟩⟩∈r{0.*1}^* using invariant_state_0_1
    by auto
  with U have ⟨⟨i,empty⟩,⟨0,starts0⟩⟩∈r{0.*1}^* using rtrancl_into_trancl2[THEN
trancl_into_rtrancl] by auto
  then have ∃q∈finalStates. ⟨⟨i,empty⟩,⟨0,q⟩⟩ ∈ r{0.*1}^* by auto
  then show thesis using DFSASatisfy_def[OF finA dfsaFrom0To1.DFSA assms(1)]
by auto
qed
```

Any list that reduces starts with 0 and ends in 1

```
theorem starts1ends0_DFSA_reduce_rev:
  fixes i
  assumes i∈Lists(2) and i {reduces in 0.*1}
  shows i0=0 and Last(i) = 1
proof-
  have finA:Finite(2) using nat_into_Finite[of 2] by auto
  have funT:transFun:states×2→states unfolding Pi_def function_def by
```

```
auto
  from assms(2) have ⟨⟨i,empty⟩,⟨0,starts0⟩⟩ ∈ r{0.*1}^*
    unfolding DFSASatisfy_def[OF finA dfsaFrom0To1.DFSA assms(1)]
    by auto
  then have ⟨⟨i,empty⟩,⟨0,starts0⟩⟩ ∈ id(field(r{0.*1})) ∪ (r{0.*1} O
r{0.*1}^*) using rtrancl_unfold[of r{0.*1}] by auto
  then have k:⟨⟨i,empty⟩,⟨0,starts0⟩⟩ ∈ id(field(r{0.*1})) ∨ ⟨⟨i,empty⟩,⟨0,starts0⟩⟩
∈ (r{0.*1} O r{0.*1}^*) by auto
  have d:domain(r{0.*1}) = NELists(2)×states
    using domainI[of _ _ r{0.*1}]
    unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
    by auto
  from k have ⟨i,empty⟩ = ⟨0,starts0⟩ ∨ ⟨⟨i,empty⟩,⟨0,starts0⟩⟩ ∈ (r{0.*1}
O r{0.*1}^*) using id_iff by auto
  then have (i=0 ∧ empty=starts0) ∨ (⟨⟨i,empty⟩,⟨0,starts0⟩⟩ ∈ r{0.*1}
O r{0.*1}^*) by auto
  then have ⟨⟨i,empty⟩,⟨0,starts0⟩⟩ ∈ r{0.*1} O r{0.*1}^* by auto
  then obtain q where q:⟨q,⟨0,starts0⟩⟩ ∈ r{0.*1} ⟨⟨i,empty⟩,q⟩∈r{0.*1}^*
unfolding comp_def by auto
  from q(1) have q ∈ domain(r{0.*1}) unfolding domain_def by auto
  with d have q ∈ NELists(2)×states by auto
  then obtain q1 q2 where qq:q=⟨q1,q2⟩ q1∈NELists(2) q2∈states by auto
  from q(1) qq(1) have A:0 = Init(q1) 0 = transFun⟨q2,Last(q1)⟩
    unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA] by
auto
  from qq(1) q(2) have restrict(i,domain(q1)) = q1
    using dfsaFrom0To1.seq_is_restriction
    by auto
  then have iRes:restrict(i,domain(q1))0 = q10 by auto
  from qq(2) obtain k where k:q1:succ(k)→2 k∈nat unfolding NELists_def
by auto
  then have Init(q1):k→2 using init_props(1) by auto
  with A(1) have k=0 unfolding Pi_def by auto
  with k(1) have qfun:q1:1→2 by auto
  from qfun have q10:Last(q1) = q10 unfolding Last_def using domain_of_fun
by auto
  from k have 0∈domain(q1) using domain_of_fun empty_in_every_succ by
auto
  with iRes have i0 = q10 using restrict by auto
  from qfun have q10∈2 using apply_type[of q1 1 λ_. 2] empty_in_every_succ[OF
nat_0I] by auto
  with qq(3) A(2) have ⟨⟨q2,Last(q1)⟩,0⟩ ∈ transFun using apply_Pair[OF
funT, of ⟨q2,Last(q1)⟩] q10 by auto
  then have q2∈2 Last(q1) = 0 by auto
  with `0∈domain(q1)` iRes q10 show i0 = 0 using restrict by auto
  from q(2) qq(1) have ⟨⟨i,empty⟩,⟨q1,q2⟩⟩ ∈ r{0.*1}^* by auto
  then have ⟨⟨i,2⟩,⟨q1,q2⟩⟩ ∈ id(field(r{0.*1})) ∪ (r{0.*1}^* O r{0.*1})
using rtrancl_rev by auto
  moreover
```

```
  {
    assume ⟨⟨i,2⟩,⟨q1,q2⟩⟩ ∈ id(field(r{0.*1}))
    then have i=q1 ∧ q2=2 by auto
    with `q2∈2` have False by auto
  } ultimately
  have ⟨⟨i,2⟩,⟨q1,q2⟩⟩ ∈ (r{0.*1}^* O r{0.*1}) by auto
  then obtain z where z:⟨⟨i,2⟩,z⟩ :r{0.*1} ⟨z,⟨q1,q2⟩⟩:r{0.*1}^* unfold-
ing comp_def by auto
  from z(2) have z∈field(r{0.*1}) using rtrancl_type[of r{0.*1}] by auto
  then obtain z1 z2 where zz:z=⟨z1,z2⟩ z1∈Lists(2) z2∈states using dfsaFrom0To1.reduce_fiel
    by blast
  from zz(1) z(1) have zzz:z1=Init(i) z2=transFun⟨2,Last(i)⟩
    unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA] by
auto
  {
    assume Last(i) = 0
    with zzz(2) have z2 = succ(2) using apply_equality[OF _ funT] by
auto
    with zz(1) z(2) have q2= succ(2) using invariant_state_3 by auto
    with `q2∈2` have False by auto
  }
  with z(1) show Last(i) = 1 using last_type[of i 2]
    unfolding DFSAExecutionRelation_def[OF finA dfsaFrom0To1.DFSA]
    by auto
qed
```

We conclude that this example constitutes the language of binary strings starting in 0 and ending in 1

```
theorem determine_strings:
  shows dfsaFrom0To1.LanguageDFSA = {i∈Lists(2). i0 = 0 ∧ Last(i) = 1}
  using starts1ends0_DFSA_reduce_rev
  starts1ends0_DFSA_reduce by blast

end
```

We define the languages determined by a deterministic finite state automaton as **regular**.

```
definition
  IsRegularLanguage (_{is a regular language on}_) where
  Finite(Σ) ⟹ L{is a regular language on}Σ ≡ ∃S s t F. ((S,s,t,F){is
an DFSA for alphabet}Σ) ∧ L=DetFinStateAuto.LanguageDFSA(S,s,t,F,Σ)
```

By definition, the language in the locale is regular.

```
corollary (in DetFinStateAuto) regular_intersect:
  shows LanguageDFSA{is a regular language on}Σ
  using IsRegularLanguage_def finite_alphabet DFSA by auto
```

A regular language is a language.

```
lemma regular_is_language:
  assumes Finite(Σ)
  and L{is a regular language on}Σ
  shows L{is a language with alphabet}Σ unfolding IsALanguage_def[OF
assms(1)]
proof-
  from assms(2) obtain S s t F where (S,s,t,F){is an DFSA for alphabet}Σ
L=DetFinStateAuto.LanguageDFSA(S,s,t,F,Σ)
    unfolding IsRegularLanguage_def[OF assms(1)] by auto
  then show L⊆ Lists(Σ)
    unfolding DetFinStateAuto_def using assms(1) by auto
qed
```

21.3 Operations on regular languages

The intersection of two regular languages is a regular language.

```
theorem regular_intersect:
  assumes Finite(Σ)
  and L1{is a regular language on}Σ
  and L2{is a regular language on}Σ
shows (L1∩L2) {is a regular language on}Σ
proof-
  from assms(1,2) obtain S1 s1 t1 F1 where l1:(S1,s1,t1,F1){is an DFSA
for alphabet}Σ
    L1 = DetFinStateAuto.LanguageDFSA(S1,s1,t1,F1,Σ)
    using IsRegularLanguage_def by auto
  then have l1:(S1,s1,t1,F1){is an DFSA for alphabet}Σ
    L1 = {i∈Lists(Σ). i <-D (S1,s1,t1,F1){in alphabet}Σ}
    using DetFinStateAuto_def assms(1) l1(1) by auto
  from assms(1,3) obtain S2 s2 t2 F2 where l2:(S2,s2,t2,F2){is an DFSA
for alphabet}Σ
    L2= DetFinStateAuto.LanguageDFSA(S2,s2,t2,F2,Σ)
    using IsRegularLanguage_def by auto
  then have l2:(S2,s2,t2,F2){is an DFSA for alphabet}Σ
    L2 = {i∈Lists(Σ). i <-D (S2,s2,t2,F2){in alphabet}Σ}
    using DetFinStateAuto_def assms(1) l2(1) by auto
  let S = S1×S2
  let s = ⟨s1,s2⟩
  let t = {⟨⟨⟨x1,x2⟩,y⟩,⟨t1⟨x1,y⟩,t2⟨x2,y⟩⟩⟩. ⟨⟨x1,x2⟩,y⟩ ∈ S×Σ}
  let F = F1×F2
  let r = DetFinStateAuto.r_D(S,s,t,Σ)
  let r1 = DetFinStateAuto.r_D(S1,s1,t1,Σ)
  let r2 = DetFinStateAuto.r_D(S2,s2,t2,Σ)
  have D:(S,s,t,F){is an DFSA for alphabet}Σ
  proof-
    have A:s∈S using l1(1) l2(1) unfolding DFSA_def[OF assms(1)] by auto
    have B:F ⊆ S using l1(1) l2(1) unfolding DFSA_def[OF assms(1)] by
auto
    have function(t) unfolding function_def by auto
```

```
    moreover have S×Σ ⊆ domain(t) by auto
    moreover
    {
      fix x1 x2 y assume as:⟨⟨x1,x2⟩,y⟩ ∈ S × Σ
      then have t1⟨x1,y⟩∈S1 t2⟨x2,y⟩∈S2 using apply_type
        l1(1) l2(1) unfolding DFSA_def[OF assms(1)] by auto
      then have ⟨t1⟨x1,y⟩,t2⟨x2,y⟩⟩∈S by auto
    }
    then have {⟨⟨⟨x1,x2⟩,y⟩,⟨t1⟨x1,y⟩,t2⟨x2,y⟩⟩⟩. ⟨⟨x1,x2⟩,y⟩ ∈ (S1 × S2)
× Σ} ∈ Pow((S×Σ)×S) by auto
    ultimately have C:t:S×Σ → S unfolding Pi_def by auto
    have Finite(S) using Finite1_L12[of S1 S2] Fin_into_Finite Finite_into_Fin
      l1(1) l2(1) unfolding DFSA_def[OF assms(1)] by auto
    with A B C show thesis unfolding DFSA_def[OF assms(1)] by auto
  qed
  then have DFSA0:DetFinStateAuto(S,s,t,F,Σ) unfolding DetFinStateAuto_def
using assms(1) by auto
  have RR:⋀m yy zz. m∈NELists(Σ) ⟹ ⟨⟨m,s1,s2⟩,⟨0,yy,zz⟩⟩:r^* ⟹ ⟨⟨m,s1⟩,⟨0,yy⟩⟩:r1^*
∧ ⟨⟨m,s2⟩,⟨0,zz⟩⟩:r2^*
    proof
      fix m yy zz
      assume as:⟨⟨m,s1,s2⟩,⟨0,yy,zz⟩⟩:r^* m∈NELists(Σ)
      note as(2)
      moreover have s1∈S1 s2∈S2 using l1(1) l2(1) unfolding DFSA_def[OF
assms(1)] by auto
      ultimately have ⟨m,s1⟩ : field(r1) ⟨m,s2⟩ : field(r2) using DetFinStateAuto.reduce_fiel
S1 s1 t1 F1 Σ] assms(1) l1(1) l2(1)
        DetFinStateAuto.reduce_field(2)[of S2 s2 t2 F2 Σ] unfolding DetFinStateAuto_def
by auto
      then have ⟨⟨m,s1⟩,⟨m,s1⟩⟩:r1^* ⟨⟨m,s2⟩,⟨m,s2⟩⟩:r2^* using rtrancl_refl
by auto moreover
      {
        fix bb cc assume as:⟨⟨m,s1,s2⟩,bb⟩:r^* ⟨bb,cc⟩:r ⟨⟨m,s1⟩,fst(bb),fst(snd(bb))⟩:r1^*
∧ ⟨⟨m,s2⟩,fst(bb),snd(snd(bb))⟩:r2^*
        from this(2) have bb∈field(r) cc∈field(r) using
            fieldI1[of bb cc r] fieldI2[of bb cc r] by auto
        then obtain bbL bb1 bb2 ccL cc1 cc2 where bbcc:bb=⟨bbL,bb1,bb2⟩
cc=⟨ccL,cc1,cc2⟩
          bbL∈Lists(Σ) ccL∈Lists(Σ) cc1∈S1 cc2∈S2 bb1∈S1 bb2∈S2
          using DetFinStateAuto.reduce_field(1)[OF DFSA0] by blast
        with as(3) have ⟨⟨m,s1⟩,bbL,bb1⟩:r1^* by auto
        from as(2) bbcc have C:ccL=Init(bbL) ⟨cc1,cc2⟩=t⟨⟨bb1,bb2⟩,Last(bbL)⟩
bbL:NELists(Σ)
          unfolding DFSAExecutionRelation_def[OF assms(1) D] by auto
        from this(2,3) bbcc(8,7) have ⟨cc1,cc2⟩=⟨t1⟨bb1,Last(bbL)⟩,t2⟨bb2,Last(bbL)⟩⟩
using last_type[of bbL Σ]
          apply_equality[of ⟨⟨bb1,bb2⟩,Last(bbL)⟩ _ t] D
          unfolding DFSA_def[OF assms(1)] by auto
        then have cc1=t1⟨bb1,Last(bbL)⟩ cc2=t2⟨bb2,Last(bbL)⟩ by auto
```

```
        with C(1,3) bbcc(8,7) have ⟨⟨bbL,bb1⟩,⟨ccL,cc1⟩⟩:r1 ⟨⟨bbL,bb2⟩,⟨ccL,cc2⟩⟩:r2
          unfolding DFSAExecutionRelation_def[OF assms(1) l1(1)]
              DFSAExecutionRelation_def[OF assms(1) l2(1)] by auto
        with as(3) bbcc(1,2) have ⟨⟨m,s1⟩,⟨ccL,cc1⟩⟩∈r1^* ⟨⟨m,s2⟩,⟨ccL,cc2⟩⟩∈r2^*
          using rtrancl_into_trancl1[of _ _ r2, THEN trancl_into_rtrancl]
          rtrancl_into_trancl1[of _ _ r1, THEN trancl_into_rtrancl]
          by auto
        then have ⟨⟨m,s1⟩,⟨fst(cc),fst(snd(cc))⟩⟩:r1^* ∧ ⟨⟨m,s2⟩,⟨fst(cc),snd(snd(cc))⟩⟩:r2^*
          using bbcc(2) by auto
      } moreover note as
      rtrancl_induct[of ⟨m,s1,s2⟩ ⟨0,yy,zz⟩ r λb. ⟨⟨m,s1⟩,⟨fst(b),fst(snd(b))⟩⟩:r1^*
∧ ⟨⟨m,s2⟩,⟨fst(b),snd(snd(b))⟩⟩:r2^*]
      ultimately show ⟨⟨m, s1⟩, 0, yy⟩ ∈ r1^* ⟨⟨m, s2⟩, 0, zz⟩ ∈ r2^* by
auto
    qed
  {
    fix m assume m∈{i∈Lists(Σ). i <-D (S,s,t,F){in alphabet}Σ}
    then have M:m∈Lists(Σ) m <-D (S,s,t,F){in alphabet}Σ by auto
    {
      assume m0:m=0
      from m0 M have 0 <-D (S,s,t,F){in alphabet}Σ by auto
      then obtain yy zz where ⟨⟨0,s1,s2⟩,⟨0,yy,zz⟩⟩:r^* ∨ s:F ⟨yy,zz⟩:F
using m0
        DFSASatisfy_def[OF assms(1) D M(1)] by auto moreover
      {
        fix y z
        assume ⟨⟨0,s1,s2⟩,y⟩ ∈ r^* ⟨y,z⟩ ∈ r y = ⟨0,s1,s2⟩
        from this(2,3) have 0∈NELists(Σ) unfolding DFSAExecutionRelation_def[OF
assms(1) D]
          by auto
        then have False unfolding NELists_def Pi_def by auto
        then have z=⟨0,s1,s2⟩ by auto
      }
      ultimately have s:F using rtrancl_induct[of ⟨0,s⟩ ⟨0,yy,zz⟩ r
        λq. q=⟨0,s⟩] by auto
      then have m <-D (S1,s1,t1,F1){in alphabet}Σ m <-D (S2,s2,t2,F2){in
alphabet}Σ
        using DFSASatisfy_def[OF assms(1) l1(1) M(1)] DFSASatisfy_def[OF
assms(1) l2(1) M(1)]
        using m0 by auto
    } moreover
    {
      assume m0:m≠0
      with M(2) obtain yy zz where F:⟨⟨m,s1,s2⟩,⟨0,yy,zz⟩⟩:r^* ⟨yy,zz⟩:F
using
        DFSASatisfy_def[OF assms(1) D M(1)] by auto
      from m0 M(1) have m∈NELists(Σ) using non_zero_List_func_is_NEList
by auto
      then have RR:⋀yy zz. ⟨⟨m,s1,s2⟩,⟨0,yy,zz⟩⟩:r^* ⟹ ⟨⟨m,s1⟩,⟨0,yy⟩⟩:r1^*
```

```
∧ ⟨⟨m,s2⟩,⟨0,zz⟩⟩:r2^*
        using RR by auto
      with F have ∃q∈F1. ⟨⟨m, s1⟩, 0,q⟩∈r1^* ∃q∈F2. ⟨⟨m, s2⟩, 0,q⟩∈r2^*
by auto
      then have m <-D (S1,s1,t1,F1){in alphabet}Σ m <-D (S2,s2,t2,F2){in
alphabet}Σ
        using DFSASatisfy_def[OF assms(1) l1(1) M(1)] DFSASatisfy_def[OF
assms(1) l2(1) M(1)]
        using m0 by auto
    }
    ultimately have m <-D (S1,s1,t1,F1){in alphabet}Σ m <-D (S2,s2,t2,F2){in
alphabet}Σ by auto
  }
  then have S1:{i ∈ Lists(Σ). i <-D (S,s,t,F){in alphabet}Σ} ⊆ L1∩L2
using l1(2) l2(2) by auto
  {
    fix m assume m∈L1∩L2
    with l1(2) l2(2) have M:m∈Lists(Σ) m <-D (S1,s1,t1,F1){in alphabet}Σ
m <-D (S2,s2,t2,F2){in alphabet}Σ
      by auto
    then obtain f1 f2 where ff:f1∈F1 f2∈F2 ⟨⟨m,s1⟩,0,f1⟩∈r1^* ∨ (m=0
∧ s1∈F1) ⟨⟨m,s2⟩,0,f2⟩∈r2^* ∨ (m=0 ∧ s2∈F2)
      unfolding DFSASatisfy_def[OF assms(1) l1(1) M(1)] DFSASatisfy_def[OF
assms(1) l2(1) M(1)] by auto
    {
      fix y z
      assume ⟨⟨0,s1⟩,y⟩ ∈ r1^* ⟨y,z⟩ ∈ r1 y = ⟨0,s1⟩
      from this(2,3) have 0∈NELists(Σ) unfolding DFSAExecutionRelation_def[OF
assms(1) l1(1)]
        by auto
      then have False unfolding NELists_def Pi_def by auto
      then have z=⟨0,s1⟩ by auto
    }
    with ff(1,3) have m=0 ⟶ s1∈F1 using rtrancl_induct[of ⟨m,s1⟩ ⟨0,f1⟩
r1 λq. q=⟨0,s1⟩] by auto
    moreover
    {
      fix y z
      assume ⟨⟨0,s2⟩,y⟩ ∈ r2^* ⟨y,z⟩ ∈ r2 y = ⟨0,s2⟩
      from this(2,3) have 0∈NELists(Σ) unfolding DFSAExecutionRelation_def[OF
assms(1) l2(1)]
        by auto
      then have False unfolding NELists_def Pi_def by auto
      then have z=⟨0,s2⟩ by auto
    }
    with ff(2,4) have m=0 ⟶ s2∈F2 using rtrancl_induct[of ⟨m,s2⟩ ⟨0,f2⟩
r2 λq. q=⟨0,s2⟩] by auto
    moreover
    {
```

```
      assume m0:m≠0
      with ff(3,4) have A:⟨⟨m,s1⟩,0,f1⟩:r1^* ⟨⟨m,s2⟩,0,f2⟩:r2^* by auto
      from this(1) have ⟨m,s1⟩:field(r1) using rtrancl_type by auto
      then have ⟨m,s1⟩:Lists(Σ)×S1 using DetFinStateAuto.reduce_field(1)[of
S1 s1 t1 F1 Σ] l1(1) unfolding DetFinStateAuto_def
        using assms(1) by auto
      with m0 have m∈NELists(Σ) s1∈S1 s2∈S2 using l1(1) l2(1) non_zero_List_func_is_NEList
        unfolding DFSA_def[OF assms(1)] by auto
      then have ⟨m,s1,s2⟩:NELists(Σ)×S by auto
      then have ⟨m,s1,s2⟩:field(r) using DetFinStateAuto.reduce_field(2)[OF
DFSA0] by auto
      then have K:⟨⟨m,s1,s2⟩,⟨m,s1,s2⟩⟩:r^* using rtrancl_refl by auto
      with `s2∈S2` have ∃f2∈S2. ⟨⟨m,s1,s2⟩,⟨m,s1,f2⟩⟩:r^* by auto more-
over
      {
        fix y z
        assume as:⟨⟨m,s1⟩,y⟩:r1^* ⟨y,z⟩:r1 ∃f2∈S2. ⟨⟨m,s1,s2⟩,⟨fst(y),snd(y),f2⟩⟩:r^*
        from as(2) obtain yL y1 where y:y=⟨yL,y1⟩ z=⟨Init(yL),t1⟨y1,Last(yL)⟩⟩
          yL∈NELists(Σ) y1:S1 unfolding DFSAExecutionRelation_def[OF
assms(1) l1(1)] by auto
        with as(3) obtain ff2 where sf:ff2∈S2 ⟨⟨m,s1,s2⟩,⟨yL,y1,ff2⟩⟩:r^*
by auto
        from this(1) y(3,4) have ⟨⟨yL,y1,ff2⟩,⟨Init(yL),t⟨⟨y1,ff2⟩,Last(yL)⟩⟩⟩
∈ r
          unfolding DFSAExecutionRelation_def[OF assms(1) D] by auto more-
over
        have funT:t:S×Σ → S using D unfolding DFSA_def[OF assms(1)]
by auto
        with y(3,4) sf(1) have t⟨⟨y1,ff2⟩,Last(yL)⟩ =⟨t1⟨y1,Last(yL)⟩,t2⟨ff2,Last(yL)⟩⟩
          using apply_equality[of ⟨⟨y1,ff2⟩,Last(yL)⟩ _ t S×Σ λ_. S] last_type
by auto
        ultimately have ⟨⟨yL,y1,ff2⟩,⟨Init(yL),⟨t1⟨y1,Last(yL)⟩,t2⟨ff2,Last(yL)⟩⟩⟩⟩
∈ r by auto
        with y(2) have ⟨⟨yL,y1,ff2⟩,⟨fst(z),⟨snd(z),t2⟨ff2,Last(yL)⟩⟩⟩⟩
∈ r by auto
        with sf(2) have ⟨⟨m,s1,s2⟩,⟨fst(z),⟨snd(z),t2⟨ff2,Last(yL)⟩⟩⟩⟩:r^*
using
          rtrancl_into_rtrancl by auto
        moreover from sf(1) have t2⟨ff2,Last(yL)⟩ ∈ S2 using l2(1)
          apply_type[of t2 S2×Σ λ_. S2] last_type[OF y(3)] unfolding
DFSA_def[OF assms(1)]
          by auto
        ultimately have ∃f2∈S2. ⟨⟨m,s1,s2⟩,⟨fst(z),⟨snd(z),f2⟩⟩⟩:r^*
          using sf(1) by auto
      } moreover note A(1)
      ultimately have ∃f2∈S2. ⟨⟨m,s1,s2⟩,⟨0,⟨f1,f2⟩⟩⟩:r^*
        using rtrancl_induct[of ⟨m,s1⟩ ⟨0,f1⟩ r1 λq. ∃f2∈S2. ⟨⟨m,s1,s2⟩,⟨fst(q),snd(q),f2⟩⟩:r
        by auto
      then obtain uu where uu:uu∈S2 ⟨⟨m,s1,s2⟩,⟨0,⟨f1,uu⟩⟩⟩:r^* by auto
```

from K `s1∈S1` have ∃f1∈S1. ⟨⟨m,s1,s2⟩,⟨m,f1,s2⟩⟩:r^* by auto moreover
{
fix y z
assume as:⟨⟨m,s2⟩,y⟩:r2^* ⟨y,z⟩:r2 ∃f1∈S1. ⟨⟨m,s1,s2⟩,⟨fst(y),f1,snd(y)⟩⟩:r^*
from as(2) obtain yL y1 where y:y=⟨yL,y1⟩ z=⟨Init(yL),t2⟨y1,Last(yL)⟩⟩
yL∈NELists(Σ) y1:S2 unfolding DFSAExecutionRelation_def[OF assms(1) l2(1)] by auto
with as(3) obtain ff2 where sf:ff2∈S1 ⟨⟨m,s1,s2⟩,⟨yL,ff2,y1⟩⟩:r^* by auto
from this(1) y(3,4) have ⟨⟨yL,ff2,y1⟩,⟨Init(yL),t⟨⟨ff2,y1⟩,Last(yL)⟩⟩⟩ ∈ r
unfolding DFSAExecutionRelation_def[OF assms(1) D] by auto moreover
have funT:t:S×Σ → S using D unfolding DFSA_def[OF assms(1)] by auto
with y(3,4) sf(1) have t⟨⟨ff2,y1⟩,Last(yL)⟩ =⟨t1⟨ff2,Last(yL)⟩,t2⟨y1,Last(yL)⟩⟩
using apply_equality[of ⟨⟨ff2,y1⟩,Last(yL)⟩ _ t S×Σ λ_. S] last_type by auto
ultimately have ⟨⟨yL,ff2,y1⟩,⟨Init(yL),⟨t1⟨ff2,Last(yL)⟩,t2⟨y1,Last(yL)⟩⟩⟩⟩ ∈ r by auto
with y(2) have ⟨⟨yL,ff2,y1⟩,⟨fst(z),⟨t1⟨ff2,Last(yL)⟩,snd(z)⟩⟩⟩ ∈ r by auto
with sf(2) have ⟨⟨m,s1,s2⟩,⟨fst(z),⟨t1⟨ff2,Last(yL)⟩,snd(z)⟩⟩⟩:r^* using
rtrancl_into_rtrancl by auto
moreover from sf(1) have t1⟨ff2,Last(yL)⟩ ∈ S1 using l1(1)
apply_type[of t1 S1×Σ λ_. S1] last_type[OF y(3)] unfolding DFSA_def[OF assms(1)]
by auto
ultimately have ∃f2∈S1. ⟨⟨m,s1,s2⟩,⟨fst(z),⟨f2,snd(z)⟩⟩⟩:r^*
using sf(1) by auto
} moreover note A(2)
ultimately have ∃f1∈S1. ⟨⟨m,s1,s2⟩,⟨0,⟨f1,f2⟩⟩⟩:r^*
using rtrancl_induct[of ⟨m,s2⟩ ⟨0,f2⟩ r2 λq. ∃f1∈S1. ⟨⟨m,s1,s2⟩,⟨fst(q),f1,snd(q)⟩⟩:r
by auto
then obtain vv where vv:vv∈S1 ⟨⟨m,s1,s2⟩,⟨0,⟨vv,f2⟩⟩⟩:r^* by auto
from uu(2) vv(2) have f1=vv uu=f2 using DetFinStateAuto.relation_deteministic[OF DFSA0,
of m s 0] by auto
from this(1) vv(2) ff(1,2) have m <-D (S,s,t,F){in alphabet}Σ
unfolding DFSASatisfy_def[OF assms(1) D M(1)] by auto
}
ultimately have m <-D (S,s,t,F){in alphabet}Σ
unfolding DFSASatisfy_def[OF assms(1) D M(1)] by auto
}
with l1(2) l2(2) have L1∩L2 ⊆ {i ∈ Lists(Σ). i <-D (S,s,t,F){in alphabet}Σ} by auto
with S1 have {i ∈ Lists(Σ). i <-D (S,s,t,F){in alphabet}Σ} = L1∩L2

```
by auto
  then have L1∩L2 = DetFinStateAuto.LanguageDFSA(S,s,t,F,Σ) by auto
  with D have ∃S s t F. ((S,s,t,F){is an DFSA for alphabet}Σ) ∧ L1∩L2
= DetFinStateAuto.LanguageDFSA(S,s,t,F,Σ)
    using exI[of λh. ((S,s,t,h){is an DFSA for alphabet}Σ) ∧ L1∩L2 =
DetFinStateAuto.LanguageDFSA(S,s,t,h,Σ) F]
    using exI[of λm. ∃h. ((S,s,m,h){is an DFSA for alphabet}Σ) ∧ L1∩L2
= DetFinStateAuto.LanguageDFSA(S,s,m,h,Σ) t]
    using exI[of λn. ∃m h. ((S,n,m,h){is an DFSA for alphabet}Σ) ∧ L1∩L2
= DetFinStateAuto.LanguageDFSA(S,n,m,h,Σ) s]
    using exI[of λp. ∃n m h. ((p,n,m,h){is an DFSA for alphabet}Σ) ∧
L1∩L2 = DetFinStateAuto.LanguageDFSA(p,n,m,h,Σ) S]
    by auto
  with assms(2,3) show thesis unfolding IsRegularLanguage_def[OF assms(1)]
IsALanguage_def[OF assms(1)] by auto
qed
```

The complement of a regular language is a regular language.

```
theorem regular_opp:
  assumes Finite(Σ)
  and L{is a regular language on}Σ
  shows (Lists(Σ)-L) {is a regular language on}Σ
proof-
  from assms(1,2) obtain S s t F where l:(S,s,t,F){is an DFSA for alphabet}Σ

    L=DetFinStateAuto.LanguageDFSA(S,s,t,F,Σ) unfolding IsRegularLanguage_def[OF
assms(1)] by auto
  then have l:(S,s,t,F){is an DFSA for alphabet}Σ
    L={i∈Lists(Σ). i <-D (S,s,t,F){in alphabet}Σ}
    using DetFinStateAuto_def assms(1) l(1) by auto
  have Lists(Σ)-L ⊆ Lists(Σ) by auto
  then have (Lists(Σ)-L) {is a language with alphabet} Σ
    unfolding IsALanguage_def[OF assms(1)] by auto
  let F = S-F
  let r = DetFinStateAuto.r_D(S,s,t,Σ)
  from l(1) have D:(S,s,t,F){is an DFSA for alphabet}Σ unfolding DFSA_def[OF
assms(1)]
    by auto
  with assms(1) have D0:DetFinStateAuto(S,s,t,F,Σ) unfolding DetFinStateAuto_def
by auto
  {
    fix m assume m∈{i∈Lists(Σ). i <-D (S,s,t,F){in alphabet}Σ}
    then have M:m∈Lists(Σ) m <-D (S,s,t,F){in alphabet}Σ by auto
    {
      assume m∈L
      with l(2) have MM:m <-D (S,s,t,F){in alphabet}Σ by auto
      {
        assume as:m=0
        from MM(1) as(1) obtain yy where ⟨⟨0,s⟩,⟨0,yy⟩⟩:r^* ∨ s:F yy∈F
```

```
unfolding
          DFSASatisfy_def[OF assms(1) l(1) M(1)] by auto moreover
        {
          fix y z
          assume ⟨⟨0,s⟩,y⟩ ∈ r^* ⟨y,z⟩ ∈ r y = ⟨0,s⟩
          from this(2,3) have 0∈NELists(Σ) unfolding DFSAExecutionRelation_def[OF
assms(1) l(1)]
            by auto
          then have False unfolding NELists_def Pi_def by auto
          then have z=⟨0,s⟩ by auto
        }
        ultimately have sf:s:F using rtrancl_induct[of ⟨0,s⟩ ⟨0,yy⟩ r λq.
q=⟨0,s⟩] by auto
        from M(2) as(1) obtain yy where ⟨⟨0,s⟩,⟨0,yy⟩⟩:r^* ∨ s:S-F yy∈S-F
unfolding
          DFSASatisfy_def[OF assms(1) D M(1)] by auto moreover
        {
          fix y z
          assume ⟨⟨0,s⟩,y⟩ ∈ r^* ⟨y,z⟩ ∈ r y = ⟨0,s⟩
          from this(2,3) have 0∈NELists(Σ) unfolding DFSAExecutionRelation_def[OF
assms(1) D]
            by auto
          then have False unfolding NELists_def Pi_def by auto
          then have z=⟨0,s⟩ by auto
        }
        ultimately have s:S-F using rtrancl_induct[of ⟨0,s⟩ ⟨0,yy⟩ r λq.
q=⟨0,s⟩] by auto
        with sf have False by auto
      }
      then have m0:m≠0 by auto
      with MM obtain q1 where q1:q1∈F ⟨⟨m, s⟩, 0, q1⟩ ∈ r^*
        unfolding DFSASatisfy_def[OF assms(1) l(1) M(1)] by auto
      from m0 M(2) obtain q2 where q2:q2∈S-F ⟨⟨m, s⟩, 0, q2⟩ ∈ r^*
        unfolding DFSASatisfy_def[OF assms(1) D M(1)] by auto
      from q1(2) q2(2) have q1=q2 using DetFinStateAuto.relation_deteministic[OF
D0,
            of m s 0] by auto
      with q1(1) q2(1) have False by auto
    }
    then have m∈Lists(Σ) - L using M(1) by auto
  }
  then have S:{i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ} ⊆ Lists(Σ)
- L by auto
  {
    fix m assume m∈Lists(Σ)-L
    then have m:m∈Lists(Σ) m <-D (S,s,t,F){in alphabet}Σ ⟹ False us-
ing l(2)
      by auto
    from this(1) have R:m = 0 ∨ (∃q∈S. ⟨⟨m,s⟩,0,q⟩ ∈ r^*)
```

```
      using non_zero_List_func_is_NEList
        DetFinStateAuto.endpoint_exists[OF D0] by auto
    {
      assume as:m=0 s∈F
      with m(1) have m <-D (S,s,t,F){in alphabet}Σ unfolding
        DFSASatisfy_def[OF assms(1) l(1) m(1)] by auto
      with m(2) have False by auto
      then have m∈{i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto
    } moreover
    {
      assume as:m=0 s∉F
      then have m=0 s∈S-F using DetFinStateAuto.DFSA_dest(1)[OF D0] by
auto
      then have m <-D (S,s,t,S - F){in alphabet}Σ unfolding DFSASatisfy_def[OF
assms(1) D m(1)] by auto
      with m(1) have m∈{i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto
    } ultimately
    have m =0 ⟶ m∈{i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto moreover
    {
      assume ∃q∈S. ⟨⟨m,s⟩,0,q⟩ ∈ r^*
      then obtain q where q:⟨⟨m,s⟩,0,q⟩ ∈ r^* q∈S by auto
      {
        assume q∈F
        with q(1) have m <-D (S,s,t,F){in alphabet}Σ unfolding DFSASatisfy_def[OF
assms(1) l(1) m(1)] by auto
        with m(2) have False by auto
      }
      with q have ∃q∈S-F. ⟨⟨m,s⟩,0,q⟩ ∈ r^* by auto
      then have m <-D (S,s,t,S-F){in alphabet}Σ unfolding DFSASatisfy_def[OF
assms(1) D m(1)] by auto
      with m(1) have m∈{i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto
    } moreover note R
    ultimately have m∈{i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto
  }
  then have Lists(Σ) -L ⊆ {i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto
  with S have Lists(Σ) -L = {i ∈ Lists(Σ) . i <-D (S,s,t,S - F){in alphabet}Σ}
by auto
  then have Lists(Σ) -L = DetFinStateAuto.LanguageDFSA(S,s,t,S-F,Σ) .
  then show thesis unfolding IsRegularLanguage_def[OF assms(1)] using
D by auto
qed
```

The union of two regular languages is a regular language.

```
theorem regular_union:
  assumes Finite(Σ)
  and L1{is a regular language on}Σ
  and L2{is a regular language on}Σ
shows (L1∪L2) {is a regular language on}Σ
proof-
  have L1∪L2 = Lists(Σ)-((Lists(Σ)-L1)∩(Lists(Σ)-L2)) using regular_is_language[OF
assms(1)]
    assms(2,3) unfolding IsALanguage_def[OF assms(1)] by auto
  moreover
  have A:(Lists(Σ)-L1) {is a regular language on}Σ using regular_opp[OF
assms(1,2)].
  have B:(Lists(Σ)-L2) {is a regular language on}Σ using regular_opp[OF
assms(1,3)].
  from A B have ((Lists(Σ)-L1)∩(Lists(Σ)-L2)) {is a regular language
on}Σ using regular_intersect[OF assms(1)] by auto
  then have (Lists(Σ)-((Lists(Σ)-L1)∩(Lists(Σ)-L2))) {is a regular language
on}Σ using regular_opp[OF assms(1)] by auto
  ultimately show thesis by auto
qed
```

Another natural operation on words is concatenation, hence we can defined the concatenated language as the set of concatenations of words of one language with words of another.

```
definition concat where
  L1 {is a language with alphabet}Σ ⟹ L2 {is a language with alphabet}Σ
    ⟹ concat(L1,L2) = {Concat(w1,w2). ⟨w1,w2⟩∈L1×L2}
```

The result of concatenating two languages is a language.

```
lemma concat_language:
  assumes Finite(Σ)
  and L1 {is a language with alphabet}Σ
  and L2 {is a language with alphabet}Σ
shows concat(L1,L2) {is a language with alphabet}Σ
proof-
  {
    fix w assume w∈concat(L1,L2)
    then obtain w1 w2 where w:w=Concat(w1,w2) w1∈L1 w2∈L2 unfolding concat_def[OF
assms(2,3)]
      by auto
    from this(2,3) assms(2,3) obtain n1 n2 where n1∈nat n2∈nat w1:n1→Σ
w2:n2→Σ
      unfolding IsALanguage_def[OF assms(1)] Lists_def by blast
    then have Concat(w1,w2):n1#+n2 → Σ n1#+n2 ∈nat using concat_props(1)
by auto
    with w(1) have w∈Lists(Σ) unfolding Lists_def by auto
  }
  then show thesis unfolding IsALanguage_def[OF assms(1)] by auto
qed
```

21.4 Non-deterministic finite state automata

We have reached a point where it is not easy to realize a concatenated language of two regular languages as a regular language. Nevertheless, if we extend our instruments to allow non-determinism it is much easier.

The cost, a priori, is that our class of languages would be larger since our automata are more generic.

The non-determinism is introduced by allowing the transition function to return not just a state, but more than one or even none.

definition
  NFSA ('(_,_,_,_'){is an NFSA for alphabet}_) **where**
  Finite(Σ) ⟹ (S,s_0,t,F){is an NFSA for alphabet}Σ ≡ Finite(S) ∧ s_0∈S ∧ F⊆S ∧ t:S×Σ→Pow(S)

The transition relation is then realized by considering all possible steps the transition function returns.

definition
  NFSAExecutionRelation ({reduce N-relation} '(_,_,_'){in alphabet}_)
where
  Finite(Σ) ⟹ (S,s_0,t,F){is an NFSA for alphabet}Σ ⟹
  {reduce N-relation}(S,s_0,t){in alphabet}Σ ≡ {⟨⟨w,Q⟩,⟨Init(w),⋃{t⟨s,Last(w)⟩. s∈Q}⟩⟩. ⟨w,Q⟩∈NELists(Σ)×Pow(S)}

The full reduction is conceived as one of those possible paths reaching a final state.

definition
  NFSASatisfy (_ <-N '(_,_,_,_'){in alphabet}_) **where**
  Finite(Σ) ⟹ (S,s_0,t,F){is an NFSA for alphabet}Σ ⟹ i∈Lists(Σ) ⟹

  i <-N (S,s_0,t,F){in alphabet}Σ ≡ (∃q∈Pow(S). (q∩F≠0 ∧ ⟨⟨i,{s_0}⟩,⟨0,q⟩⟩∈ ({reduce N-relation}(S,s_0,t){in alphabet}Σ)^*)) ∨ (i = 0 ∧ s_0∈F)

An extra generalization can be consider if we allow the transition relation to go forward without consuming elements from the word. This is implemented as allowing Σ to symbolize an step without the word being touched. We might call it a Σ transition or a ε-transition.

definition
  FullNFSA ('(_,_,_,_'){is an ε-NFSA for alphabet}_) **where**
  Finite(Σ) ⟹ (S,s_0,t,F){is an ε-NFSA for alphabet}Σ ≡ Finite(S) ∧ s_0∈S ∧ F⊆S ∧ t:S×succ(Σ)→Pow(S)

The closure of a set of states can then be viewed as all the states reachable from that set with a transition of type Σ.

definition
  EpsilonClosure (ε-cl) **where**
  Finite(Σ) ⟹ (S,s_0,t,F){is an ε-NFSA for alphabet}Σ ⟹ E⊆S

$\Longrightarrow$ ε-cl(S,t,Σ,E) ≡ ⋃{P∈Pow(S). ⟨E,P⟩∈({⟨Q,{s∈S. ∃q∈Q. t⟨q,Σ⟩ = s}⟩. Q∈Pow(S)}^*)}

The reduction relation is then extended by considering any such transitions.

definition
  FullNFSAExecutionRelation ({reduce ε-N-relation} '(_,_,_')}{in alphabet}_)
where
  Finite(Σ) ⟹ (S,s_0,t,F){is an ε-NFSA for alphabet}Σ ⟹
  {reduce ε-N-relation}(S,s_0,t){in alphabet}Σ ≡ {⟨⟨w,Q⟩,⟨Init(w),ε-cl(S,t,Σ,⋃{t⟨s,Last(w)⟩. s∈Q})⟩⟩. ⟨w,Q⟩∈NELists(Σ)×Pow(S)}

The full reduction of a word is similar to that of the automata without ε-transitions.

definition
  FullNFSASatisfy (_ <-ε-N '(_,_,_,_')}{in alphabet}_) **where**
  Finite(Σ) ⟹ (S,s_0,t,F){is an ε-NFSA for alphabet}Σ ⟹ i∈Lists(Σ) ⟹
  i <-ε-N (S,s_0,t,F){in alphabet}Σ ≡ (∃q∈Pow(S). (q∩F≠0 ∧ ⟨⟨i,{s_0}⟩,⟨0,q⟩⟩∈ ({reduce ε-N-relation}(S,s_0,t){in alphabet}Σ)^*)) ∨ (i = 0 ∧ s_0∈F)

We define a locale to create some notation

locale NonDetFinStateAuto =
  fixes S **and** s_0 **and** t **and** F **and** Σ
  assumes finite_alphabet: Finite(Σ)

  assumes NFSA: (S,s_0,t,F){is an NFSA for alphabet}Σ

Notation for the transition relation

abbreviation (**in** NonDetFinStateAuto) nd_rel (r_N)
  where r_N ≡ {reduce N-relation}(S,s_0,t){in alphabet}Σ

Notation for the language generated by the non-deterministic automaton

abbreviation (**in** NonDetFinStateAuto) LanguageNFSA
  where LanguageNFSA ≡ {i∈Lists(Σ). i<-N (S,s_0,t,F){in alphabet}Σ}

21.5 Equivalence of Non-deterministic and Deterministic Finite State Automata

We will show that the non-deterministic automata generate languages that are regular in the sense that there is a deterministic automaton that generates the same language.

The transition function of the deterministic automata we will construct

definition (**in** NonDetFinStateAuto) tPow **where**
  tPow ≡ {⟨⟨U,u⟩,(⋃v∈U. t ⟨v, u⟩)⟩. ⟨U,u⟩∈Pow(S)×Σ}

The transition relation of the deterministic automata we will construct

```
definition (in NonDetFinStateAuto) rPow where
  rPow ≡ DetFinStateAuto.r_D(Pow(S),{s_0},tPow,Σ)
```

We show that we do have a deterministic automaton

```
sublocale NonDetFinStateAuto < dfsa:DetFinStateAuto Pow(S) {s_0} tPow {Q∈Pow(S).
Q∩F ≠ 0} Σ
  unfolding DetFinStateAuto_def DFSA_def[OF finite_alphabet] unfolding
tPow_def
  apply safe using finite_alphabet apply simp
  using NFSA unfolding NFSA_def[OF finite_alphabet]
  apply simp using NFSA unfolding NFSA_def[OF finite_alphabet]
   apply simp unfolding Pi_def function_def apply auto
proof-
  fix b y x v assume as:y ∈ Σ b ⊆ S v ∈ b x ∈ t  ⟨v, y⟩
  from as(2,3) have v:v∈S by auto
  have t ∈ S × Σ → Pow(S) using NFSA
    unfolding NFSA_def[OF finite_alphabet] by auto
  with as(1,4) v show x∈S using apply_type[of t S×Σ λ_. Pow(S) ⟨v,y⟩]
    by auto
qed
```

The two automata have the same relations associated with them.

First, we show that if the non-deterministic automaton produces a reduction step to a word, then the deterministic one we constructed does the same reduction step.

```
lemma (in NonDetFinStateAuto) nd_impl_det:
  assumes ⟨⟨w,Q⟩,⟨u,G⟩⟩∈r_N
  shows ⟨⟨w,Q⟩,⟨u,G⟩⟩∈rPow
proof-
  from assms have w:w∈NELists(Σ) u=Init(w) Q∈Pow(S) G=(⋃s∈Q. t⟨s,Last(w)⟩)
    unfolding NFSAExecutionRelation_def[OF finite_alphabet NFSA] by auto
  then have tPow⟨Q,Last(w)⟩ = (⋃s∈Q. t⟨s,Last(w)⟩) ⟹ thesis
    unfolding DFSAExecutionRelation_def[OF finite_alphabet dfsa.DFSA]
rPow_def
    by auto
  moreover have ⟨⟨Q,Last(w)⟩,⋃s∈Q. t⟨s,Last(w)⟩⟩:tPow using last_type[OF
w(1)] w(3)
    unfolding tPow_def by auto
  ultimately show thesis using apply_equality[OF _ dfsa.DFSA_dest(3),
of ⟨Q,Last(w)⟩ ⋃s∈Q. t⟨s,Last(w)⟩]
    by blast
qed
```

Next, we show that if the deterministic automaton produces a reduction step to a word, then the non-deterministic one we constructed does the same reduction step.

```
lemma (in NonDetFinStateAuto) det_impl_nd:
```

```
  assumes ⟨⟨w,Q⟩,⟨u,G⟩⟩∈rPow
  shows ⟨⟨w,Q⟩,⟨u,G⟩⟩∈r_N
proof-
  from assms have w:w∈NELists(Σ) u=Init(w) Q∈Pow(S) G=tPow ⟨Q, Last(w)⟩
    unfolding DFSAExecutionRelation_def[OF finite_alphabet dfsa.DFSA]
rPow_def by auto
  then have tPow⟨Q,Last(w)⟩ = (⋃s∈Q. t⟨s,Last(w)⟩) ⟹ thesis
    unfolding NFSAExecutionRelation_def[OF finite_alphabet NFSA] by auto
  moreover have ⟨⟨Q,Last(w)⟩,⋃s∈Q. t⟨s,Last(w)⟩⟩:tPow unfolding tPow_def
using last_type[OF w(1)] w(3) by auto
  ultimately show thesis using apply_equality[OF _ dfsa.DFSA_dest(3),
of ⟨Q,Last(w)⟩ ⋃s∈Q. t⟨s,Last(w)⟩]
    by blast
qed
```

Since both are relations, they are equal

```
corollary (in NonDetFinStateAuto) relation_NFSA_to_DFSA:
  shows r_N = rPow using nd_impl_det det_impl_nd
  unfolding DFSAExecutionRelation_def[OF finite_alphabet dfsa.DFSA]
  NFSAExecutionRelation_def[OF finite_alphabet NFSA] rPow_def
  by auto
```

As a consequence, by the definition of a language generated by an automaton, both languages are equal.

```
theorem (in NonDetFinStateAuto) language_nfsa:
  shows dfsa.LanguageDFSA = LanguageNFSA
proof-
  let S = Pow(S)
  let s = {s_0}
  let f = {⟨⟨U, u⟩, ⋃v∈U. t  ⟨v, u⟩⟩ . ⟨U, u⟩ ∈ Pow(S) × Σ}
  let F = {Q ∈ Pow(S) . Q ∩ F ≠ 0}
  {
    fix i assume i:i∈Lists(Σ) i <-D (S,s,f,F){in alphabet}Σ
    {
      assume i=0 s∈F
      then have i=0 s_0∈F by auto
      then have i <-N (S,s_0,t,F){in alphabet}Σ
        unfolding NFSASatisfy_def[OF finite_alphabet NFSA i(1)] by auto
    } moreover
    {
      assume ¬(i=0 ∧ s∈F)
      with i(2) obtain q where q:q∈F ⟨⟨i,s⟩,⟨0,q⟩⟩∈rPow^*
        using DFSASatisfy_def[OF finite_alphabet dfsa.DFSA i(1)]
        unfolding rPow_def tPow_def by auto
      then have ⟨⟨i,s⟩,⟨0,q⟩⟩∈r_N^* using relation_NFSA_to_DFSA
        by auto
      with q(1) have i <-N (S,s_0,t,F){in alphabet}Σ
      unfolding NFSASatisfy_def[OF finite_alphabet NFSA i(1)] by auto
    } ultimately
```

```
    have i <-N (S,s₀,t,F){in alphabet}Σ by auto
  }
  then have A:{i ∈ Lists(Σ) . dfsa.reduce(i)} ⊆ {i ∈ Lists(Σ) . i <-N
(S,s₀,t,F){in alphabet}Σ}
    unfolding rPow_def tPow_def by auto
  {
    fix i assume i:i∈Lists(Σ) i <-N (S,s₀,t,F){in alphabet}Σ
    {
      assume i=0 s₀∈F
      then have i=0 s∈F using NFSA
        unfolding NFSA_def[OF finite_alphabet] by auto
      then have i <-D (S,s,f,F){in alphabet}Σ
        using DFSASatisfy_def[OF finite_alphabet dfsa.DFSA i(1)]
        unfolding tPow_def rPow_def by auto
    } moreover
    {
      assume ¬(i=0 ∧ s₀∈F)
      with i(2) obtain q where q:q∈Pow(S) q∩F≠0 ⟨⟨i,s⟩,⟨0,q⟩⟩∈r_N^*
        unfolding NFSASatisfy_def[OF finite_alphabet NFSA i(1)] by auto
      then have ⟨⟨i,s⟩,⟨0,q⟩⟩∈rPow^* using relation_NFSA_to_DFSA
        by auto
      with q(1,2) have i <-D (S,s,f,F){in alphabet}Σ
        using DFSASatisfy_def[OF finite_alphabet dfsa.DFSA i(1)]
      unfolding tPow_def rPow_def by auto
    } ultimately
    have i <-D (S,s,f,F){in alphabet}Σ by auto
  }
  then have B:{i ∈ Lists(Σ) . i <-N (S,s₀,t,F){in alphabet}Σ}⊆
    {i ∈ Lists(Σ) . dfsa.reduce(i)} unfolding tPow_def rPow_def by auto
  with A show dfsa.LanguageDFSA = LanguageNFSA by auto
qed
```

The language of a non-deterministic finite state automaton is regular.

```
corollary (in NonDetFinStateAuto) lang_is_regular:
  shows LanguageNFSA{is a regular language on}Σ
  unfolding IsRegularLanguage_def[OF finite_alphabet]
  apply (rule exI[of _ Pow(S)],
         rule exI[of _ {s₀}],
         rule exI[of _ tPow],
         rule exI[of _ {Q ∈ Pow(S). Q ∩ F ≠ 0}])
  using language_nfsa dfsa.DFSA by auto
```

```
end
```

22 Inductive sequences

```
theory InductiveSeq_ZF imports Nat_ZF_IML FiniteSeq_ZF FinOrd_ZF
```

begin

In this theory we discuss sequences defined by conditions of the form $a_0 = x,\ a_{n+1} = f(a_n)$ and similar.

22.1 Sequences defined by induction

One way of defining a sequence (that is a function $a : \mathbb{N} \to X$) is to provide the first element of the sequence and a function to find the next value when we have the current one. This is usually called "defining a sequence by induction". In this section we set up the notion of a sequence defined by induction and prove the theorems needed to use it.

First we define a helper notion of the sequence defined inductively up to a given natural number n.

```
definition
  InductiveSequenceN(x,f,n) ≡
  THE a. a: succ(n) → domain(f) ∧ a(0) = x ∧ (∀k∈n. a(succ(k)) = f(a(k)))
```

From that we define the inductive sequence on the whole set of natural numbers. Recall that in Isabelle/ZF the set of natural numbers is denoted `nat`.

```
definition
  InductiveSequence(x,f) ≡ ⋃n∈nat. InductiveSequenceN(x,f,n)
```

First we will consider the question of existence and uniqueness of finite inductive sequences. The proof is by induction and the next lemma is the $P(0)$ step. To understand the notation recall that for natural numbers in set theory we have $n = \{0, 1, .., n-1\}$ and `succ(n)`$= \{0, 1, .., n\}$.

```
lemma indseq_exun0: assumes A1: f: X→X and A2: x∈X
  shows
  ∃! a. a: succ(0) → X ∧ a(0) = x ∧ ( ∀k∈0. a(succ(k)) = f(a(k)) )
proof
  fix a b
  assume A3:
    a: succ(0) → X ∧ a(0) = x ∧ ( ∀k∈0. a(succ(k)) = f(a(k)) )
    b: succ(0) → X ∧ b(0) = x ∧ ( ∀k∈0. b(succ(k)) = f(b(k)) )
  moreover have succ(0) = {0} by auto
  ultimately have a: {0} → X  b: {0} → X by auto
  then have a = {⟨0, a(0)⟩}   b = {⟨0, b(0)⟩} using func_singleton_pair
    by auto
  with A3 show a=b by simp
next
  let a = {⟨0,x⟩}
  have a : {0} → {x} using singleton_fun by simp
  moreover from A1 A2 have {x} ⊆ X by simp
```

```
  ultimately have a : {0} → X
    using func1_1_L1B by blast
  moreover have {0} = succ(0) by auto
  ultimately have a : succ(0) → X by simp
  with A1 show
    ∃ a. a: succ(0) → X ∧ a(0) = x ∧ (∀k∈0. a(succ(k)) = f(a(k)))
    using singleton_apply by auto
qed
```

A lemma about restricting finite sequences needed for the proof of the inductive step of the existence and uniqueness of finite inductive seqences.

```
lemma indseq_restrict:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat and
  A4: a: succ(succ(n))→ X ∧ a(0) = x ∧ (∀k∈succ(n). a(succ(k)) = f(a(k)))
  and A5: a_r = restrict(a,succ(n))
  shows
  a_r: succ(n) → X ∧ a_r(0) = x ∧ ( ∀k∈n. a_r(succ(k)) = f(a_r(k)) )
proof -
  from A3 have succ(n) ⊆ succ(succ(n)) by auto
  with A4 A5 have a_r: succ(n) → X using restrict_type2 by auto
  moreover
  from A3 have 0 ∈ succ(n) using empty_in_every_succ by simp
  with A4 A5 have a_r(0) = x using restrict_if by simp
  moreover from A3 A4 A5 have ∀k∈n. a_r(succ(k)) = f(a_r(k))
    using succ_ineq restrict_if by auto
  ultimately show thesis by simp
qed
```

Existence and uniqueness of finite inductive sequences. The proof is by induction and the next lemma is the inductive step.

```
lemma indseq_exun_ind:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat and
  A4: ∃! a. a: succ(n) → X ∧ a(0) = x ∧ (∀k∈n. a(succ(k)) = f(a(k)))
  shows
  ∃! a. a: succ(succ(n)) → X ∧ a(0) = x ∧
  ( ∀k∈succ(n). a(succ(k)) = f(a(k)) )
proof
  fix a b assume
    A5: a: succ(succ(n)) → X ∧ a(0) = x ∧
    ( ∀k∈succ(n). a(succ(k)) = f(a(k)) ) and
    A6: b: succ(succ(n)) → X ∧ b(0) = x ∧
    ( ∀k∈succ(n). b(succ(k)) = f(b(k)) )
  show a = b
  proof -
    let a_r = restrict(a,succ(n))
    let b_r = restrict(b,succ(n))
    note A1 A2 A3 A5
    moreover have a_r = restrict(a,succ(n)) by simp
    ultimately have I:
```

```
      a_r: succ(n) → X ∧ a_r(0) = x ∧ ( ∀k∈n. a_r(succ(k)) = f(a_r(k)) )
      by (rule indseq_restrict)
    note A1 A2 A3 A6
    moreover have b_r = restrict(b,succ(n)) by simp
    ultimately have
      b_r: succ(n) → X ∧ b_r(0) = x ∧ ( ∀k∈n. b_r(succ(k)) = f(b_r(k)) )
      by (rule indseq_restrict)
    with A4 I have II: a_r = b_r by blast
    from A3 have succ(n) ∈ nat by simp
    moreover from A5 A6 have
      a: succ(succ(n)) → X and b: succ(succ(n)) → X
      by auto
    moreover note II
    moreover
    have T: n ∈ succ(n) by simp
    then have a_r(n) = a(n) and b_r(n) = b(n) using restrict
      by auto
    with A5 A6 II T have a(succ(n)) = b(succ(n)) by simp
    ultimately show a = b by (rule finseq_restr_eq)
  qed
next show
    ∃ a. a: succ(succ(n)) → X ∧ a(0) = x ∧
    ( ∀k∈succ(n). a(succ(k)) = f(a(k)) )
  proof -
    from A4 obtain a where III: a: succ(n) → X and IV: a(0) = x
      and V: ∀k∈n. a(succ(k)) = f(a(k)) by auto
    let b = a ∪ {⟨succ(n), f(a(n))⟩}
    from A1 III have
      VI: b : succ(succ(n)) → X and
      VII: ∀k ∈ succ(n). b(k) = a(k) and
      VIII: b(succ(n)) = f(a(n))
      using apply_funtype finseq_extend by auto
    from A3 have 0 ∈ succ(n) using empty_in_every_succ by simp
    with IV VII have IX: b(0) = x by auto
    { fix k assume k ∈ succ(n)
      then have k∈n ∨ k = n by auto
      moreover
      { assume A7: k ∈ n
 with A3 VII have b(succ(k)) = a(succ(k))
   using succ_ineq by auto
 also from A7 V VII have a(succ(k)) = f(b(k)) by simp
 finally have b(succ(k)) =  f(b(k)) by simp }
      moreover
      { assume A8: k = n
 with VIII have b(succ(k)) =  f(a(k)) by simp
 with A8 VII VIII have b(succ(k)) =  f(b(k)) by simp }
      ultimately have b(succ(k)) =  f(b(k)) by auto
    } then have ∀k ∈ succ(n). b(succ(k)) =  f(b(k)) by simp
    with VI IX show thesis by auto
```

```
  qed
qed
```

The next lemma combines `indseq_exun0` and `indseq_exun_ind` to show the existence and uniqueness of finite sequences defined by induction.

```
lemma indseq_exun:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat
  shows
  ∃! a. a: succ(n) → X ∧ a(0) = x ∧ (∀k∈n. a(succ(k)) = f(a(k)))
proof -
  note A3
  moreover from A1 A2 have
    ∃! a. a: succ(0) → X ∧ a(0) = x ∧ ( ∀k∈0. a(succ(k)) = f(a(k)) )
    using indseq_exun0 by simp
  moreover from A1 A2 have ∀k ∈ nat.
    ( ∃! a. a: succ(k) → X ∧ a(0) = x ∧
    ( ∀i∈k. a(succ(i)) = f(a(i)) )) ⟶
    ( ∃! a. a: succ(succ(k)) → X ∧ a(0) = x ∧
    ( ∀i∈succ(k). a(succ(i)) = f(a(i)) ) )
    using indseq_exun_ind by simp
  ultimately show
    ∃! a. a: succ(n) → X ∧ a(0) = x ∧ ( ∀k∈n. a(succ(k)) = f(a(k)) )
    by (rule ind_on_nat)
qed
```

We are now ready to prove the main theorem about finite inductive sequences.

```
theorem fin_indseq_props:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat and
  A4: a = InductiveSequenceN(x,f,n)
  shows
  a: succ(n) → X
  a(0) = x
  ∀k∈n. a(succ(k)) = f(a(k))
proof -
  let i = THE a. a: succ(n) → X ∧ a(0) = x ∧
    ( ∀k∈n. a(succ(k)) = f(a(k)) )
  from A1 A2 A3 have
    ∃! a. a: succ(n) → X ∧ a(0) = x ∧ ( ∀k∈n. a(succ(k)) = f(a(k)) )
    using indseq_exun by simp
  then have
    i: succ(n) → X ∧ i(0) = x ∧ ( ∀k∈n. i(succ(k)) = f(i(k)) )
    by (rule theI)
  moreover from A1 A4 have a = i
    using InductiveSequenceN_def func1_1_L1 by simp
  ultimately show
    a: succ(n) → X   a(0) = x   ∀k∈n. a(succ(k)) = f(a(k))
    by auto
qed
```

Since we have uniqueness we can show the inverse of `fin_indseq_props`: a sequence that satisfies the inductive sequence properties listed there is the inductively defined sequence.

```
lemma is_fin_indseq:
  assumes n ∈ nat f: X→X x∈X and
  a: succ(n) → X a(0) = x ∀k∈n. a(succ(k)) = f(a(k))
  shows a = InductiveSequenceN(x,f,n)
proof -
  let b = InductiveSequenceN(x,f,n)
  from assms(1,2,3) have
    b: succ(n) → X b(0) = x ∀k∈n. b(succ(k)) = f(b(k))
    using fin_indseq_props by simp_all
  with assms show thesis using indseq_exun by blast
qed
```

A corollary about the domain of a finite inductive sequence.

```
corollary fin_indseq_domain:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat
  shows domain(InductiveSequenceN(x,f,n)) = succ(n)
proof -
  from assms have InductiveSequenceN(x,f,n) : succ(n) → X
    using fin_indseq_props by simp
  then show thesis using func1_1_L1 by simp
qed
```

The collection of finite sequences defined by induction is consistent in the sense that the restriction of the sequence defined on a larger set to the smaller set is the same as the sequence defined on the smaller set.

```
lemma indseq_consistent: assumes A1: f: X→X and A2: x∈X and
  A3: i ∈ nat  j ∈ nat and A4: i ⊆ j
  shows
  restrict(InductiveSequenceN(x,f,j),succ(i)) = InductiveSequenceN(x,f,i)
proof -
  let a = InductiveSequenceN(x,f,j)
  let b = restrict(InductiveSequenceN(x,f,j),succ(i))
  let c = InductiveSequenceN(x,f,i)
  from A1 A2 A3 have
    a: succ(j) → X  a(0) = x   ∀k∈j. a(succ(k)) = f(a(k))
    using fin_indseq_props by auto
  with A3 A4 have
    b: succ(i) → X ∧ b(0) = x ∧ ( ∀k∈i. b(succ(k)) = f(b(k)))
    using succ_subset restrict_type2 empty_in_every_succ restrict succ_ineq
    by auto
  moreover from A1 A2 A3 have
    c: succ(i) → X ∧ c(0) = x ∧ ( ∀k∈i. c(succ(k)) = f(c(k)))
    using fin_indseq_props by simp
  moreover from A1 A2 A3 have
    ∃! a. a: succ(i) → X ∧ a(0) = x ∧ ( ∀k∈i. a(succ(k)) = f(a(k)) )
```

```
    using indseq_exun by simp
  ultimately show b = c by blast
qed
```

For any two natural numbers one of the corresponding inductive sequences is contained in the other.

```
lemma indseq_subsets: assumes A1: f: X→X and A2: x∈X and
  A3: i ∈ nat  j ∈ nat and
  A4: a = InductiveSequenceN(x,f,i)  b = InductiveSequenceN(x,f,j)
  shows a ⊆ b ∨ b ⊆ a
proof -
  from A3 have i⊆j ∨ j⊆i using nat_incl_total by simp
  moreover
  { assume i⊆j
    with A1 A2 A3 A4 have restrict(b,succ(i)) = a
      using indseq_consistent by simp
    moreover have restrict(b,succ(i)) ⊆ b
      using restrict_subset by simp
    ultimately have a ⊆ b ∨ b ⊆ a by simp }
  moreover
  { assume j⊆i
    with A1 A2 A3 A4 have restrict(a,succ(j)) = b
      using indseq_consistent by simp
    moreover have restrict(a,succ(j)) ⊆ a
      using restrict_subset by simp
    ultimately have a ⊆ b ∨ b ⊆ a by simp }
  ultimately show  a ⊆ b ∨ b ⊆ a by auto
qed
```

The inductive sequence generated by applying a function 0 times is just the singleton list containing the starting point.

```
lemma indseq_empty: assumes f: X→X x∈X
  shows
    InductiveSequenceN(x,f,0):{0}→X
    InductiveSequenceN(x,f,0) = {⟨0,x⟩}
proof -
  let a = InductiveSequenceN(x,f,0)
  from assms have a:succ(0)→X and a(0) = x
    using fin_indseq_props(1,2) by simp_all
  moreover have succ(0) = {0} by auto
  ultimately show a:{0}→X by auto
  then have a = {⟨0,a(0)⟩} using func_singleton_pair
    by simp
  with‹a(0) = x› show a = {⟨0,x⟩} by simp
qed
```

The tail of an inductive sequence generated by f and started from x is the same as the inductive sequence started from $f(x)$.

```
lemma indseq_tail: assumes n ∈ nat f: X→X x∈X
```

```
  shows Tail(InductiveSequenceN(x,f,succ(n))) = InductiveSequenceN(f(x),f,n)
proof -
  let a = Tail(InductiveSequenceN(x,f,succ(n)))
  from assms(2,3) have f(x)∈X using apply_funtype by simp
  have  a: succ(n) → X a(0) = f(x) and
        ∀k∈n. a(succ(k)) = f(a(k))
  proof -
    let b = InductiveSequenceN(x,f,succ(n))
    from assms have I: succ(n)∈nat b: succ(succ(n)) → X
      using fin_indseq_props(1) by simp_all
    then show Tail(b):succ(n)→X using tail_props by simp
    from assms(1) I have II: Tail(b)(0) = b(succ(0))
      using tail_props empty_in_every_succ by blast
    from assms ‹succ(n)∈nat› have b(succ(0)) = f(b(0))
      using fin_indseq_props(3) empty_in_every_succ by blast
    moreover from assms(2,3) ‹succ(n)∈nat› have b(0) = x
      using fin_indseq_props(2) by simp
    ultimately have b(succ(0)) = f(x) by simp
    with II show a(0) = f(x) by simp
    { fix k assume k∈n
      from I have III: ∀k∈succ(n). a(k) = b(succ(k))
        using tail_props by blast
      with assms(1) ‹k∈n› have a(succ(k)) = b(succ(succ(k)))
        using succ_ineq by blast
      with assms ‹k∈n› III have a(succ(k)) = f(a(k))
        using succ_ineq fin_indseq_props(3) by simp
    } then show ∀k∈n. a(succ(k)) = f(a(k))
      by simp
  qed
  with assms(1,2) ‹f(x)∈X› show thesis by (rule is_fin_indseq)
qed
```

The first theorem about properties of infinite inductive sequences: inductive sequence is a indeed a sequence (i.e. a function on the set of natural numbers.

```
theorem indseq_seq: assumes  A1: f: X→X and A2: x∈X
  shows InductiveSequence(x,f) : nat → X
proof -
  let S = {InductiveSequenceN(x,f,n). n ∈ nat}
  { fix a assume a∈S
    then obtain n where n ∈ nat and a =  InductiveSequenceN(x,f,n)
      by auto
    with A1 A2 have a : succ(n)→X using fin_indseq_props
      by simp
    then have ∃A B. a:A→B by auto
  } then have ∀a ∈ S. ∃A B. a:A→B by auto
  moreover
  { fix a b assume a∈S  b∈S
    then obtain i j where i∈nat  j∈nat and
      a = InductiveSequenceN(x,f,i)   b = InductiveSequenceN(x,f,j)
```

```
      by auto
    with A1 A2 have a⊆b ∨ b⊆a using indseq_subsets by simp
  } then have ∀a∈S. ∀b∈S. a⊆b ∨ b⊆a by auto
  ultimately have ⋃S : domain(⋃S) → range(⋃S)
    using fun_Union by simp
  with A1 A2 have I: ⋃S : nat → range(⋃S)
    using domain_UN fin_indseq_domain nat_union_succ by simp
  moreover
  { fix k assume A3: k ∈ nat
    let y = (⋃S)(k)
    note I A3
    moreover have y = (⋃S)(k) by simp
    ultimately have ⟨k,y⟩ ∈ (⋃S) by (rule func1_1_L5A)
    then obtain n where n ∈ nat and II: ⟨k,y⟩ ∈ InductiveSequenceN(x,f,n)
      by auto
    with A1 A2 have InductiveSequenceN(x,f,n): succ(n) → X
      using fin_indseq_props by simp
    with II have y ∈ X using func1_1_L5 by blast
  } then have ∀k ∈ nat.  (⋃S)(k) ∈ X by simp
  ultimately have ⋃S : nat → X using func1_1_L1A
    by blast
  then show InductiveSequence(x,f) : nat → X
    using InductiveSequence_def by simp
qed
```

Restriction of an inductive sequence to a finite domain is the corresponding finite inductive sequence.

```
lemma indseq_restr_eq:
  assumes A1: f: X→X and A2: x∈X and A3: n ∈ nat
  shows
  restrict(InductiveSequence(x,f),succ(n)) = InductiveSequenceN(x,f,n)
proof -
  let a = InductiveSequence(x,f)
  let b = InductiveSequenceN(x,f,n)
  let S = {InductiveSequenceN(x,f,n). n ∈ nat}
  from A1 A2 A3 have
    I: a : nat → X  and succ(n) ⊆ nat
    using indseq_seq succnat_subset_nat by auto
  then have restrict(a,succ(n)) : succ(n) → X
    using restrict_type2 by simp
  moreover from A1 A2 A3 have b : succ(n) → X
    using fin_indseq_props by simp
  moreover
  { fix k assume A4: k ∈ succ(n)
    from A1 A2 A3 I have
      ⋃S : nat → X   b ∈ S  b : succ(n) → X
      using InductiveSequence_def fin_indseq_props by auto
    with A4 have restrict(a,succ(n))(k) = b(k)
      using fun_Union_apply InductiveSequence_def restrict_if
```

```
      by simp
  } then have ∀k ∈ succ(n). restrict(a,succ(n))(k) = b(k)
    by simp
  ultimately show thesis by (rule func_eq)
qed
```

The first element of the inductive sequence starting at x and generated by f is indeed x.

```
theorem indseq_valat0: assumes A1: f: X→X and A2: x∈X
  shows InductiveSequence(x,f)(0) = x
proof -
  let a = InductiveSequence(x,f)
  let b = InductiveSequenceN(x,f,0)
  have T: 0∈nat  0 ∈ succ(0) by auto
  with A1 A2 have b(0) = x
    using fin_indseq_props by simp
  moreover from T have restrict(a,succ(0))(0) = a(0)
    using restrict_if by simp
  moreover from A1 A2 T have
    restrict(a,succ(0)) = b
    using indseq_restr_eq by simp
  ultimately show a(0) = x by simp
qed
```

An infinite inductive sequence satisfies the inductive relation that defines it.

```
theorem indseq_vals:
  assumes A1: f: X→X and A2: x∈X  and A3: n ∈ nat
  shows
  InductiveSequence(x,f)(succ(n)) = f(InductiveSequence(x,f)(n))
proof -
  let a = InductiveSequence(x,f)
  let b = InductiveSequenceN(x,f,succ(n))
  from A3 have T:
    succ(n) ∈ succ(succ(n))
    succ(succ(n)) ∈ nat
    n ∈ succ(succ(n))
    by auto
  then have a(succ(n)) = restrict(a,succ(succ(n)))(succ(n))
    using restrict_if by simp
  also from A1 A2 T have ... = f(restrict(a,succ(succ(n)))(n))
    using indseq_restr_eq fin_indseq_props by simp
  also from T have ... = f(a(n)) using restrict_if by simp
  finally show a(succ(n)) = f(a(n)) by simp
qed
```

22.2 Images of inductive sequences

In this section we consider the properties of sets that are images of inductive sequences, that is are of the form $\{f^{(n)}(x) : n \in N\}$ for some x in the domain of f, where $f^{(n)}$ denotes the n'th iteration of the function f. For a function $f : X \to X$ and a point $x \in X$ such set is set is sometimes called the orbit of x generated by f.

The basic properties of orbits.

```
theorem ind_seq_image: assumes A1: f: X→X and A2: x∈X  and
  A3: A = InductiveSequence(x,f)(nat)
  shows x∈A and ∀y∈A. f(y) ∈ A
proof -
  let a = InductiveSequence(x,f)
  from A1 A2 have a : nat → X using indseq_seq
    by simp
  with A3 have I: A = {a(n). n ∈ nat} using func_imagedef
    by auto hence a(0) ∈ A by auto
  with A1 A2 show x∈A using indseq_valat0 by simp
  { fix y assume y∈A
    with I obtain n where II: n ∈ nat and III: y = a(n)
      by auto
    with A1 A2 have a(succ(n)) = f(y)
      using indseq_vals by simp
    moreover from I II have a(succ(n)) ∈ A by auto
    ultimately have f(y) ∈ A by simp
  } then show ∀y∈A. f(y) ∈ A by simp
qed
```

22.3 Subsets generated by a binary operation

In algebra we often talk about sets "generated" by an element, that is sets of the form (in multiplicative notation) $\{a^n | n \in Z\}$. This is a related to a general notion of "power" (as in $a^n = a \cdot a \cdot .. \cdot a$) or multiplicity $n \cdot a = a + a + .. + a$. The intuitive meaning of such notions is obvious, but we need to do some work to be able to use it in the formalized setting. This sections is devoted to sequences that are created by repeatedly applying a binary operation with the second argument fixed to some constant.

Basic propertes of sets generated by binary operations.

```
theorem binop_gen_set:
  assumes A1: f: X×Y → X and A2: x∈X  y∈Y and
  A3: a = InductiveSequence(x,Fix2ndVar(f,y))
  shows
  a : nat → X
  a(nat) ∈ Pow(X)
  x ∈ a(nat)
  ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat)
proof -
  let g = Fix2ndVar(f,y)
  from A1 A2 have I: g : X→X
    using fix_2nd_var_fun by simp
```

```
  with A2 A3 show a : nat → X
    using indseq_seq by simp
  then show a(nat) ∈ Pow(X) using func1_1_L6 by simp
  from A2 A3 I show x ∈ a(nat) using ind_seq_image by blast
  from A2 A3 I have
    g : X→X  x∈X  a(nat) = InductiveSequence(x,g)(nat)
    by auto
  then show ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat)
    by (rule ind_seq_image)
qed
```

A simple corollary to the theorem binop_gen_set: a set that contains all iterations of the application of a binary operation exists.

```
lemma binop_gen_set_ex: assumes A1: f: X×Y → X and A2: x∈X  y∈Y
  shows {A ∈ Pow(X). x∈A ∧ (∀z ∈ A. f⟨z,y⟩ ∈ A) } ≠ 0
proof -
  let a = InductiveSequence(x,Fix2ndVar(f,y))
  let A = a(nat)
  from A1 A2 have I: A ∈ Pow(X) and x ∈ A using binop_gen_set
    by auto
  moreover
  { fix z assume T: z∈A
    with A1 A2 have Fix2ndVar(f,y)(z) ∈ A
      using binop_gen_set by simp
    moreover
    from I T have z ∈ X by auto
    with A1 A2 have Fix2ndVar(f,y)(z) = f⟨z,y⟩
      using fix_var_val by simp
    ultimately have f⟨z,y⟩ ∈ A by simp
  } then have ∀z ∈ A. f⟨z,y⟩ ∈ A by simp
  ultimately show thesis by auto
qed
```

A more general version of binop_gen_set where the generating binary operation acts on a larger set.

```
theorem binop_gen_set1: assumes A1: f: X×Y → X and
  A2: X₁ ⊆ X and A3: x∈X₁  y∈Y and
  A4: ∀t∈X₁. f⟨t,y⟩ ∈ X₁ and
  A5: a = InductiveSequence(x,Fix2ndVar(restrict(f,X₁×Y),y))
shows
  a : nat → X₁
  a(nat) ∈ Pow(X₁)
  x ∈ a(nat)
  ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat)
  ∀z ∈ a(nat). f⟨z,y⟩ ∈ a(nat)
proof -
  let h = restrict(f,X₁×Y)
  let g = Fix2ndVar(h,y)
  from A2 have X₁×Y ⊆ X×Y by auto
```

```
  with A1 have I: h : X₁×Y → X
    using restrict_type2 by simp
  with A3 have II: g: X₁ → X using fix_2nd_var_fun by simp
  from A3 A4 I have ∀t∈X₁. g(t) ∈ X₁
    using restrict fix_var_val by simp
  with II have III: g : X₁ → X₁ using func1_1_L1A by blast
  with A3 A5 show a : nat → X₁ using indseq_seq by simp
  then show IV: a(nat) ∈ Pow(X₁) using func1_1_L6 by simp
  from A3 A5 III show x ∈ a(nat) using ind_seq_image by blast
  from A3 A5 III have
    g : X₁ → X₁   x∈X₁  a(nat) =  InductiveSequence(x,g)(nat)
    by auto
  then have ∀z ∈ a(nat). Fix2ndVar(h,y)(z) ∈ a(nat)
    by (rule ind_seq_image)
  moreover
  { fix z assume z ∈ a(nat)
    with IV have z ∈ X₁ by auto
    with A1 A2 A3 have g(z) = Fix2ndVar(f,y)(z)
      using fix_2nd_var_restr_comm restrict by simp
  } then have ∀z ∈ a(nat). g(z) = Fix2ndVar(f,y)(z) by simp
  ultimately show ∀z ∈ a(nat). Fix2ndVar(f,y)(z) ∈ a(nat) by simp
  moreover
  { fix z assume z ∈ a(nat)
    with A2 IV have z∈X by auto
    with A1 A3 have Fix2ndVar(f,y)(z) = f⟨z,y⟩
      using fix_var_val by simp
  } then have ∀z ∈ a(nat). Fix2ndVar(f,y)(z) = f⟨z,y⟩
    by simp
  ultimately show ∀z ∈ a(nat). f⟨z,y⟩ ∈ a(nat)
    by simp
qed
```

A generalization of binop_gen_set_ex that applies when the binary operation acts on a larger set. This is used in our Metamath translation to prove the existence of the set of real natural numbers. Metamath defines the real natural numbers as the smallest set that cantains 1 and is closed with respect to operation of adding 1.

```
lemma binop_gen_set_ex1: assumes A1: f: X×Y → X and
  A2: X₁ ⊆ X and A3: x∈X₁  y∈Y and
  A4: ∀t∈X₁. f⟨t,y⟩ ∈ X₁
  shows {A ∈ Pow(X₁). x∈A ∧ (∀z ∈ A. f⟨z,y⟩ ∈ A) } ≠ 0
proof -
  let a = InductiveSequence(x,Fix2ndVar(restrict(f,X₁×Y),y))
  let A = a(nat)
  from A1 A2 A3 A4 have
    A ∈ Pow(X₁)   x ∈ A   ∀z ∈ A. f⟨z,y⟩ ∈ A
    using binop_gen_set1 by auto
  thus thesis by auto
qed
```

22.4 Inductive sequences with changing generating function

A seemingly more general form of a sequence defined by induction is a sequence generated by the difference equation $x_{n+1} = f_n(x_n)$ where $n \mapsto f_n$ is a given sequence of functions such that each maps X into inself. For example when $f_n(x) := x + x_n$ then the equation $S_{n+1} = f_n(S_n)$ describes the sequence $n \mapsto S_n = s_0 + \sum_{i=0}^{n} x_n$, i.e. the sequence of partial sums of the sequence $\{s_0, x_0, x_1, x_3, ..\}$.

The situation where the function that we iterate changes with n can be derived from the simpler case if we define the generating function appropriately. Namely, we replace the generating function in the definitions of `InductiveSequenceN` by the function $f : X \times n \to X \times n$, $f\langle x, k\rangle = \langle f_k(x), k+1\rangle$ if $k < n$, $\langle f_k(x), k\rangle$ otherwise. The first notion defines the expression we will use to define the generating function. To understand the notation recall that in standard Isabelle/ZF for a pair $s = \langle x, n\rangle$ we have $\texttt{fst}(s) = x$ and $\texttt{snd}(s) = n$.

definition
```
  StateTransfFunNMeta(F,n,s) ≡
  if (snd(s) ∈ n) then ⟨F(snd(s))(fst(s)), succ(snd(s))⟩ else s
```

Then we define the actual generating function on sets of pairs from $X \times \{0, 1, .., n\}$.

definition
```
  StateTransfFunN(X,F,n) ≡ {⟨s, StateTransfFunNMeta(F,n,s)⟩. s ∈ X×succ(n)}
```

Having the generating function we can define the expression that we cen use to define the inductive sequence generates.

definition
```
  StatesSeq(x,X,F,n) ≡
  InductiveSequenceN(⟨x,0⟩, StateTransfFunN(X,F,n),n)
```

Finally we can define the sequence given by a initial point x, and a sequence F of n functions.

definition
```
  InductiveSeqVarFN(x,X,F,n) ≡ {⟨k,fst(StatesSeq(x,X,F,n)(k))⟩. k ∈ succ(n)}
```

The state transformation function (`StateTransfFunN` is a function that transforms $X \times n$ into itself.

```
lemma state_trans_fun: assumes A1: n ∈ nat and A2: F: n → (X→X)
  shows StateTransfFunN(X,F,n): X×succ(n) → X×succ(n)
proof -
  { fix s assume A3: s ∈ X×succ(n)
    let x = fst(s)
    let k = snd(s)
    let S = StateTransfFunNMeta(F,n,s)
```

```
    from A3 have T: x ∈ X  k ∈ succ(n) and ⟨x,k⟩ = s by auto
    { assume A4: k ∈ n
      with A1 have succ(k) ∈ succ(n) using succ_ineq by simp
      with A2 T A4 have S ∈ X×succ(n)
 using apply_funtype StateTransfFunNMeta_def by simp }
    with A2 A3 T have S ∈ X×succ(n)
      using apply_funtype StateTransfFunNMeta_def by auto
  } then have ∀s ∈ X×succ(n). StateTransfFunNMeta(F,n,s) ∈ X×succ(n)
    by simp
  then have
    {⟨s, StateTransfFunNMeta(F,n,s)⟩. s ∈ X×succ(n)} : X×succ(n) → X×succ(n)
    by (rule ZF_fun_from_total)
  then show StateTransfFunN(X,F,n): X×succ(n) → X×succ(n)
    using StateTransfFunN_def by simp
qed
```

We can apply `fin_indseq_props` to the sequence used in the definition of `InductiveSeqVarFN` to get the properties of the sequence of states generated by the `StateTransfFunN`.

```
lemma states_seq_props:
  assumes A1: n ∈ nat and A2: F: n → (X→X) and A3: x∈X and
  A4: b = StatesSeq(x,X,F,n)
  shows
  b : succ(n) → X×succ(n)
  b(0) = ⟨x,0⟩
  ∀k ∈ succ(n). snd(b(k)) = k
  ∀k∈n. b(succ(k)) = ⟨F(k)(fst(b(k))), succ(k)⟩
proof -
  let f = StateTransfFunN(X,F,n)
  from A1 A2 have I: f : X×succ(n) → X×succ(n)
    using state_trans_fun by simp
  moreover from A1 A3 have II: ⟨x,0⟩ ∈ X×succ(n)
    using empty_in_every_succ by simp
  moreover note A1
  moreover from A4 have III: b = InductiveSequenceN(⟨x,0⟩,f,n)
    using StatesSeq_def by simp
  ultimately show IV: b : succ(n) → X×succ(n)
    by (rule fin_indseq_props)
  from I II A1 III show V: b(0) = ⟨x,0⟩
    by (rule fin_indseq_props)
  from I II A1 III have VI: ∀k∈n. b(succ(k)) = f(b(k))
    by (rule fin_indseq_props)
  { fix k
    note I
    moreover
    assume A5: k ∈ n hence k ∈ succ(n) by auto
    with IV have b(k) ∈  X×succ(n) using apply_funtype by simp
    moreover have f = {⟨s, StateTransfFunNMeta(F,n,s)⟩. s ∈ X×succ(n)}
      using StateTransfFunN_def by simp
```

```
    ultimately have f(b(k)) =  StateTransfFunNMeta(F,n,b(k))
      by (rule ZF_fun_from_tot_val)
  } then have VII: ∀k ∈ n. f(b(k)) =  StateTransfFunNMeta(F,n,b(k))
    by simp
  { fix k assume A5: k ∈ succ(n)
    note A1 A5
    moreover from V have  snd(b(0)) = 0 by simp
    moreover from VI VII have
      ∀j∈n. snd(b(j)) = j ⟶ snd(b(succ(j))) = succ(j)
      using StateTransfFunNMeta_def by auto
    ultimately have snd(b(k)) = k by (rule fin_nat_ind)
  } then show ∀k ∈ succ(n). snd(b(k)) = k by simp
  with VI VII show ∀k∈n. b(succ(k)) = ⟨F(k)(fst(b(k))), succ(k)⟩
    using StateTransfFunNMeta_def by auto
qed
```

Basic properties of sequences defined by equation $x_{n+1} = f_n(x_n)$.

```
theorem fin_indseq_var_f_props:
  assumes A1: n ∈ nat and A2: x∈X and A3: F: n → (X→X) and
  A4: a = InductiveSeqVarFN(x,X,F,n)
  shows
  a: succ(n) → X
  a(0) = x
  ∀k∈n. a(succ(k)) = F(k)(a(k))
proof -
  let f = StateTransfFunN(X,F,n)
  let b = StatesSeq(x,X,F,n)
  from A1 A2 A3 have b : succ(n) → X×succ(n)
    using states_seq_props by simp
  then have ∀k ∈ succ(n). b(k) ∈ X×succ(n)
    using apply_funtype by simp
  hence ∀k ∈ succ(n). fst(b(k)) ∈ X by auto
  then have I: {⟨k,fst(b(k))⟩. k ∈ succ(n)} : succ(n) → X
    by (rule ZF_fun_from_total)
  with A4 show II: a: succ(n) → X using InductiveSeqVarFN_def
    by simp
  moreover from A1 have 0 ∈ succ(n) using empty_in_every_succ
    by simp
  moreover from A4 have III:
    a = {⟨k,fst(StatesSeq(x,X,F,n)(k))⟩. k ∈ succ(n)}
    using InductiveSeqVarFN_def by simp
  ultimately have a(0) = fst(b(0))
    by (rule ZF_fun_from_tot_val)
  with A1 A2 A3 show a(0) = x using states_seq_props by auto
  { fix k
    assume A5: k ∈ n
    with A1 have T1: succ(k) ∈ succ(n) and T2: k ∈ succ(n)
      using succ_ineq by auto
    from II T1 III have a(succ(k)) = fst(b(succ(k)))
```

```
      by (rule ZF_fun_from_tot_val)
    with A1 A2 A3 A5 have a(succ(k)) = F(k)(fst(b(k)))
      using states_seq_props by simp
    moreover from II T2 III have a(k) = fst(b(k))
      by (rule ZF_fun_from_tot_val)
    ultimately have a(succ(k)) =  F(k)(a(k))
      by simp
  } then show ∀k∈n. a(succ(k)) = F(k)(a(k))
    by simp
qed
```

Uniqueness lemma for sequences generated by equation $x_{n+1} = f_n(x_n)$:

```
lemma fin_indseq_var_f_uniq: assumes n∈nat x∈X F: n → (X→X)
  and a: succ(n) → X a(0) = x ∀k∈n. a(succ(k)) = (F(k))(a(k))
  and b: succ(n) → X b(0) = x ∀k∈n. b(succ(k)) = (F(k))(b(k))
  shows a=b
proof -
  have ∀k∈succ(n). a(k) = b(k)
  proof -
    let A = {i∈succ(succ(n)). ∀k∈i.  a(k) = b(k)}
    let m = Maximum(Le,A)
    from assms(1) have I: succ(succ(n)) ∈ nat A⊆succ(succ(n)) by auto
    moreover
    from assms(1,5,8) have succ(0) ∈ A using empty_in_every_succ succ_ineq
      by simp
    hence II: A≠0 by auto
    ultimately have m∈A by (rule nat_max_props)
    moreover have m = succ(n)
    proof -
      { assume m ≠ succ(n)
        from I II have III: ∀k∈A. k ≤ m by (rule nat_max_props)
        have succ(m) ∈ A
        proof -
          from ‹m ≠ succ(n)› ‹m∈A› have m∈succ(n)
            using mem_succ_not_eq by blast
          from I II have m ∈ nat by (rule nat_max_props)
          from ‹succ(0) ∈ A› III have succ(0) ≤ m by blast
          hence m ≠ 0 by auto
          with ‹m ∈ nat› obtain k where k∈nat m = succ(k)
            using Nat_ZF_1_L3 by auto
          with assms(1) ‹m∈succ(n)› have k∈n using succ_mem by simp
          with assms(6,9) ‹m = succ(k)› ‹m∈A›
          have a(m) = b(m) using succ_explained by simp
          with assms(1) ‹m∈A› ‹m∈succ(n)› show succ(m) ∈ A
            using succ_explained succ_ineq by blast
        qed
        with III have succ(m) ≤ m by (rule property_holds)
        hence False by auto
      } thus thesis by auto
```

```
    qed
    ultimately show thesis by simp
  qed
  with assms(4,7) show a=b by (rule func_eq)
qed
```

A sequence that has the properties of sequences generated by equation $x_{n+1} = f_n(x_n)$ must be the one generated by this equation.

```
theorem is_fin_indseq_var_f:  assumes n∈nat x∈X F: n → (X→X)
  and a: succ(n) → X a(0) = x ∀k∈n. a(succ(k)) = (F(k))(a(k))
shows a = InductiveSeqVarFN(x,X,F,n)
proof -
  let b = InductiveSeqVarFN(x,X,F,n)
  from assms(1,2,3) have b: succ(n) → X  b(0) = x
    and ∀k∈n. b(succ(k)) = F(k)(b(k))
    using fin_indseq_var_f_props by simp_all
  with assms show thesis by (rule fin_indseq_var_f_uniq)
qed
```

A consistency condition: if we make the sequence of generating functions shorter, then we get a shorter inductive sequence with the same values as in the original sequence.

```
lemma fin_indseq_var_f_restrict: assumes
  A1: n ∈ nat  i ∈ nat  x∈X  F: n → (X→X)   G: i → (X→X)
  and A2: i ⊆ n and  A3: ∀j∈i. G(j) = F(j) and A4: k ∈ succ(i)
  shows InductiveSeqVarFN(x,X,G,i)(k) = InductiveSeqVarFN(x,X,F,n)(k)
proof -
  let a = InductiveSeqVarFN(x,X,F,n)
  let b = InductiveSeqVarFN(x,X,G,i)
  from A1 A4 have i ∈ nat  k ∈ succ(i) by auto
  moreover from A1 have b(0) = a(0)
    using fin_indseq_var_f_props by simp
  moreover from A1 A2 A3 have
    ∀j∈i. b(j) = a(j) ⟶ b(succ(j)) = a(succ(j))
    using fin_indseq_var_f_props by auto
  ultimately show b(k) = a(k)
    by (rule fin_nat_ind)
qed
```

22.5 The Pascal's triangle

One possible application of the inductive sequences is to define the Pascal's triangle. The Pascal's triangle can be defined directly as $P_{n,k} = \binom{n}{k} = \frac{n!}{k!(n-k)!}$ for $n \geq k \geq 0$. Formalizing this definition (or explaining to a 10-years old) is quite difficult as it depends on the definition of factorial and some facts about factorizing natural numbers needed to show that the quotient in $\frac{n!}{k!(n-k)!}$ is always a natural number. Another approach uses induction and the property that each number in the array is the sum of the two numbers directly above it.

To shorten the definition of the function generating the Pascal's trangle we first define expression for the k'th element in the row following given row r. The rows are represented as lists, i.e. functions $r : n \to \mathbb{N}$ (recall that for natural numbers we have $n = \{0, 1, 2, ..., n-1\}$). The value of the next row is 1 at the beginning and equals $r(k-1) + r(k)$ otherwise. A careful reader might wonder why we do not require the values to be 1 on the right boundary of the Pascal's triangle. We are able to show this as a theorem (see `binom_right_boundary` below) using the fact that in Isabelle/ZF the value of a function on an argument that is outside of the domain is the empty set, which is the same as zero of natural numbers.

```
definition
  BinomElem(r,k) ≡ if k=0 then 1 else r(pred(k)) #+ r(k)
```

Next we define a function that takes a row in a Pascal's triangle and returns the next row.

```
definition
  GenBinom ≡ {⟨r,{⟨k,BinomElem(r,k)⟩. k∈succ(domain(r))}⟩. r∈NELists(nat)}
```

The function generating rows of the Pascal's triangle is indeed a function that maps nonempty lists of natural numbers into nonempty lists of natural numbers.

```
lemma gen_binom_fun: shows GenBinom: NELists(nat) → NELists(nat)
proof -
  { fix r assume r ∈ NELists(nat)
    then obtain n where n∈nat and r:succ(n)→nat
      unfolding NELists_def by auto
    then have domain(r) = succ(n) using func1_1_L1 by simp
    let r₁ = {⟨k,BinomElem(r,k)⟩. k∈succ(domain(r))}
    have ∀k∈succ(domain(r)). BinomElem(r,k) ∈ nat
      unfolding BinomElem_def by simp
    then have r₁: succ(domain(r))→nat
      by (rule ZF_fun_from_total)
    with ‹n∈nat› ‹domain(r) = succ(n)› have r₁∈NELists(nat)
      unfolding NELists_def by auto
  } then show thesis using ZF_fun_from_total unfolding GenBinom_def
    by simp
qed
```

The value of the function `GenBinom` at a nonempty list r is a list of length one greater than the length of r.

```
lemma gen_binom_fun_val: assumes n∈nat r:succ(n)→nat
  shows GenBinom(r):succ(succ(n)) → nat
proof -
  let B = {⟨r,{⟨k,BinomElem(r,k)⟩. k∈succ(domain(r))}⟩. r∈NELists(nat)}
```

```
  let r₁ = {⟨k,BinomElem(r,k)⟩. k∈succ(domain(r))}
  from assms have r∈NELists(nat) unfolding NELists_def by blast
  then have B(r) = r₁ using ZF_fun_from_tot_val1 by simp
  have ∀k∈succ(domain(r)). BinomElem(r,k) ∈ nat
    unfolding BinomElem_def by simp
  then have r₁: succ(domain(r))→nat
    by (rule ZF_fun_from_total)
  with assms(2) ‹B(r) = r₁› show thesis
    using func1_1_L1 unfolding GenBinom_def by simp
qed
```

Now we are ready to define the Pascal's triangle as the inductive sequence that starts from a singleton list $0 \mapsto 1$ and is generated by iterations of the GenBinom function.

```
definition
  PascalTriangle ≡ InductiveSequence({⟨0,1⟩},GenBinom)
```

The singleton list containing 1 (i.e. the starting point of the inductive sequence that defines the PascalTriangle) is a finite list and the PascalTriangle is a sequence (an infinite list) of nonempty lists of natural numbers.

```
lemma pascal_sequence:
  shows {⟨0,1⟩} ∈ NELists(nat) and PascalTriangle: nat → NELists(nat)
  using list_len1_singleton(2) gen_binom_fun indseq_seq
  unfolding PascalTriangle_def
  by auto
```

The GenBinom function creates the next row of the Pascal's triangle from the previous one.

```
lemma binom_gen: assumes n∈nat
  shows PascalTriangle(succ(n)) = GenBinom(PascalTriangle(n))
  using assms pascal_sequence gen_binom_fun indseq_vals
  unfolding PascalTriangle_def by simp
```

The n'th row of the Pascal's triangle is a list of $n+1$ natural numbers.

```
lemma pascal_row_list:
  assumes n∈nat shows PascalTriangle(n):succ(n)→nat
proof -
  from assms(1) have n∈nat and PascalTriangle(0):succ(0)→nat
    using gen_binom_fun pascal_sequence(1) indseq_valat0 list_len1_singleton(1)
    unfolding PascalTriangle_def by auto
  moreover have
    ∀k∈nat. PascalTriangle(k):succ(k)→nat ⟶
      PascalTriangle(succ(k)):succ(succ(k))→nat
  proof -
    { fix k assume k∈nat and PascalTriangle(k):succ(k)→nat
      then have PascalTriangle(succ(k)):succ(succ(k))→nat
        using gen_binom_fun_val gen_binom_fun pascal_sequence(1) indseq_vals
```

```
      unfolding NELists_def PascalTriangle_def
      by auto
  } thus thesis by simp
 qed
 ultimately show thesis by (rule ind_on_nat)
qed
```

In our approach the Pascal's triangle is a list of lists. The value at index $n \in \mathbb{N}$ is a list of length $n+1$ (see `pascal_row_list` above). Hence, the largest index in the domain of this list is n. However, we can still show that the value of that list at index $n+1$ is 0, because in Isabelle/ZF (as well as in Metamath) the value of a function at a point outside of the domain is the empty set, which happens to be the same as the natural number 0.

```
lemma pascal_val_beyond: assumes n∈nat
  shows (PascalTriangle(n))(succ(n)) = 0
proof -
  from assms have domain(PascalTriangle(n)) = succ(n)
    using pascal_row_list func1_1_L1 by blast
  then show thesis using mem_self apply_0
    by simp
qed
```

For $n > 0$ the Pascal's triangle values at (n, k) are given by the `BinomElem` expression.

```
lemma pascal_row_val: assumes n∈nat k∈succ(succ(n))
  shows (PascalTriangle(succ(n)))(k) = BinomElem(PascalTriangle(n),k)
proof -
  let B = {⟨r,{⟨k,BinomElem(r,k)⟩. k∈succ(domain(r))}⟩. r∈NELists(nat)}
  let r = PascalTriangle(n)
  let B_r = {⟨k,BinomElem(r,k)⟩. k∈succ(succ(n))}
  from assms(1) have r ∈ NELists(nat) and r : succ(n)→nat
    using pascal_sequence(2) apply_funtype pascal_row_list
    by auto
  then have B(r) = B_r using func1_1_L1 ZF_fun_from_tot_val1
    by simp
  moreover from assms(1) have B(r) = PascalTriangle(succ(n))
    using binom_gen unfolding GenBinom_def by simp
  moreover from assms(2) have B_r(k) = BinomElem(r,k)
    by (rule ZF_fun_from_tot_val1)
  ultimately show thesis by simp
qed
```

The notion that will actually be used is the binomial coefficient $\binom{n}{k}$ which we define as the value at the right place of the Pascal's triangle.

```
definition
  Binom(n,k) ≡ (PascalTriangle(n))(k)
```

Entries in the Pascal's triangle are natural numbers. Since in Isabelle/ZF the value of a function at a point that is outside of the domain is the empty set (which is the same as zero of natural numbers) we do not need any assumption on k.

```
lemma binom_in_nat: assumes n∈nat shows Binom(n,k) ∈ nat
proof -
  { assume k ∈ succ(n)
    with assms have (PascalTriangle(n))(k) ∈ nat
      using pascal_row_list apply_funtype by blast
  }
  moreover
  { assume k ∉ succ(n)
    from assms have domain(PascalTriangle(n)) = succ(n)
      using pascal_row_list func1_1_L1 by blast
    with ‹k ∉ succ(n)› have (PascalTriangle(n))(k) = 0
      using apply_0 by simp
    hence (PascalTriangle(n))(k) ∈ nat by simp
  }
  ultimately show thesis unfolding Binom_def by blast
qed
```

The top of the Pascal's triangle is equal to 1 (i.e. $\binom{0}{0} = 1$). This is an easy fact that it is useful to have handy as it is at the start of a couple of inductive arguments.

```
lemma binom_zero_zero: shows Binom(0,0) = 1
  using gen_binom_fun pascal_sequence(1) indseq_valat0 pair_val
    unfolding Binom_def PascalTriangle_def by auto
```

The binomial coefficients are 1 on the left boundary of the Pascal's triangle.

```
theorem binom_left_boundary: assumes n∈nat shows Binom(n,0) = 1
proof -
  { assume n≠0
    with assms obtain k where k∈nat and n = succ(k)
      using Nat_ZF_1_L3 by blast
    then have Binom(n,0) = 1 using empty_in_every_succ pascal_row_val
      unfolding BinomElem_def Binom_def by simp
  }
  then show thesis using binom_zero_zero by blast
qed
```

The main recursive property of binomial coefficients: each number in the $\binom{n}{k}$, $n > 0, 0 \neq k \leq n$ array (i.e. the Pascal's triangle except the top) is the sum of the two numbers directly above it. The statement looks like it has an off-by-one error in the assumptions, but it's ok and needed later.

```
theorem binom_prop: assumes n∈nat k ≤ n #+ 1 k≠0
  shows Binom(n #+ 1,k) = Binom(n,k #- 1) #+ Binom(n,k)
proof -
  let P = PascalTriangle
  from assms(1,2) have k∈nat and k ∈ succ(succ(n))
```

```
    using le_in_nat nat_mem_lt(2) by auto
  with assms(1) have Binom(n #+ 1,k) = BinomElem(P(n),k)
    unfolding Binom_def using pascal_row_val by simp
  also from assms(3) ‹k∈nat› have
    BinomElem(P(n),k) = (P(n))(k #- 1) #+ (P(n))(k)
    unfolding BinomElem_def using pred_minus_one by simp
  also have (P(n))(k #- 1) #+ (P(n))(k) =  Binom(n,k #- 1) #+ Binom(n,k)
    unfolding Binom_def by simp
  finally show thesis by simp
qed
```

A version binom_prop where we write $k+1$ instead of k.

```
lemma binom_prop2: assumes n∈nat k ∈ n #+ 1
  shows Binom(n #+ 1,k #+ 1) = Binom(n,k #+ 1) #+ Binom(n,k)
proof -
  from assms have k∈nat using elem_nat_is_nat(2) by blast
  hence k #+1 #- 1 = k by simp
  moreover from assms have
    Binom(n #+ 1,k #+ 1) = Binom(n,k #+1 #- 1) #+ Binom(n,k #+ 1)
    using succ_ineq2 binom_prop by simp
  ultimately show thesis by simp
qed
```

A special case of binom_prop when $n = k + 1$ that helps with the induction step in the proof that the binomial coefficient are 1 on the right boundary of the Pascal's triangle.

```
lemma binom_prop1: assumes n∈nat
  shows Binom(n #+ 1,n #+ 1) = Binom(n,n)
proof -
  let B = Binom
  from assms have B(n,n) ∈ nat
    using pascal_row_list apply_funtype
    unfolding Binom_def by blast
  from assms have (PascalTriangle(n))(succ(n)) = 0
    using pascal_val_beyond by simp
  moreover from assms have succ(n) = n #+ 1
    using succ_add_one(1) by simp
  ultimately have B(n,n #+ 1) = 0
    unfolding Binom_def by simp
  with assms ‹B(n,n) ∈ nat› show thesis
    using succ_add_one(2) binom_prop add_subtract add_0 add_commute
    by simp
qed
```

The binomial coefficients are 1 on the right boundary of the Pascal's triangle.

```
theorem binom_right_boundary: assumes n∈nat shows Binom(n,n) = 1
proof -
  from assms have n∈nat and Binom(0,0) = 1
```

```
    using binom_zero_zero by auto
  moreover have
    ∀k∈nat. Binom(k,k) = 1 ⟶ Binom(succ(k),succ(k)) = 1
    using binom_prop1 by simp
  ultimately show thesis by (rule ind_on_nat)
qed
```

end

23 Enumerations

theory Enumeration_ZF **imports** NatOrder_ZF FiniteSeq_ZF FinOrd_ZF

begin

Suppose r is a linear order on a set A that has n elements, where $n \in \mathbb{N}$. In the FinOrd_ZF theory we prove a theorem stating that there is a unique order isomorphism between $n = \{0, 1, .., n-1\}$ (with natural order) and A. Another way of stating that is that there is a unique way of counting the elements of A in the order increasing according to relation r. Yet another way of stating the same thing is that there is a unique sorted list of elements of A. We will call this list the Enumeration of A.

23.1 Enumerations: definition and notation

In this section we introduce the notion of enumeration and define a proof context (a "locale" in Isabelle terms) that sets up the notation for writing about enumarations.

We define enumeration as the only order isomorphism beween a set A and the number of its elements. We are using the formula $\bigcup\{x\} = x$ to extract the only element from a singleton. Le is the (natural) order on natural numbers, defined is Nat_ZF theory in the standard Isabelle library.

```
definition
  Enumeration(A,r) ≡ ⋃ ord_iso(|A|,Le,A,r)
```

To set up the notation we define a locale enums. In this locale we will assume that r is a linear order on some set X. In most applications this set will be just the set of natural numbers. Standard Isabelle uses $\leq$ to denote the "less or equal" relation on natural numbers. We will use the $\leq$ symbol to denote the relation r. Those two symbols usually look the same in the presentation, but they are different in the source.To shorten the notation the enumeration Enumeration(A,r) will be denoted as $\sigma(A)$. Similarly as in the Semigroup theory we will write $a \hookleftarrow x$ for the result of appending an element

x to the finite sequence (list) a. Finally, $a \sqcup b$ will denote the concatenation of the lists a and b.

```
locale enums =

  fixes X r
  assumes linord: IsLinOrder(X,r)

  fixes ler (infix ≤ 70)
  defines ler_def[simp]: x ≤ y ≡ ⟨x,y⟩ ∈ r

  fixes σ
  defines σ_def [simp]: σ(A) ≡ Enumeration(A,r)

  fixes append (infix ↩ 72)
  defines append_def[simp]: a ↩ x ≡ Append(a,x)

  fixes concat (infixl ⊔ 69)
  defines concat_def[simp]: a ⊔ b ≡ Concat(a,b)
```

23.2 Properties of enumerations

In this section we prove basic facts about enumerations.

A special case of the existence and uniqueess of the order isomorphism for finite sets when the first set is a natural number.

```
lemma (in enums) ord_iso_nat_fin:
  assumes A ∈ FinPow(X) and n ∈ nat and A ≈ n
  shows ∃!f. f ∈ ord_iso(n,Le,A,r)
  using assms NatOrder_ZF_1_L2 linord nat_finpow_nat
    fin_ord_iso_ex_uniq by simp
```

An enumeration is an order isomorhism, a bijection, and a list.

```
lemma (in enums) enum_props: assumes A ∈ FinPow(X)
  shows
  σ(A) ∈ ord_iso(|A|,Le, A,r)
  σ(A) ∈ bij(|A|,A)
  σ(A) : |A| → A
proof -
  from assms have
    IsLinOrder(nat,Le) and |A| ∈ FinPow(nat) and  A ≈ |A|
    using NatOrder_ZF_1_L2 card_fin_is_nat nat_finpow_nat
    by auto
  with assms show σ(A) ∈ ord_iso(|A|,Le, A,r)
    using linord fin_ord_iso_ex_uniq singleton_extract
      Enumeration_def by simp
  then show σ(A) ∈ bij(|A|,A) and σ(A) : |A| → A
    using ord_iso_def bij_def surj_def
    by auto
```

qed

A corollary from `enum_props`. Could have been attached as another assertion, but this slows down verification of some other proofs.

```
lemma (in enums) enum_fun: assumes A ∈ FinPow(X)
  shows σ(A) : |A| → X
proof -
  from assms have σ(A) : |A| → A and A⊆X
    using enum_props  FinPow_def by auto
  then show σ(A) : |A| → X by (rule func1_1_L1B)
qed
```

If a list is an order isomorphism then it must be the enumeration.

```
lemma (in enums) ord_iso_enum: assumes A1: A ∈ FinPow(X) and
  A2: n ∈ nat and A3: f ∈ ord_iso(n,Le,A,r)
  shows f = σ(A)
proof -
  from A3 have n ≈ A using ord_iso_def eqpoll_def
    by auto
  then have A ≈ n by (rule eqpoll_sym)
  with A1 A2 have ∃!f. f ∈ ord_iso(n,Le,A,r)
    using ord_iso_nat_fin by simp
  with assms ‹A ≈ n› show f = σ(A)
    using enum_props card_card by blast
qed
```

What is the enumeration of the empty set?

```
lemma (in enums) empty_enum: shows σ(0) = 0
proof -
  have
    0 ∈ FinPow(X) and 0 ∈ nat and 0 ∈ ord_iso(0,Le,0,r)
    using empty_in_finpow empty_ord_iso_empty
    by auto
  then show σ(0) = 0 using ord_iso_enum
    by blast
qed
```

Adding a new maximum to a set appends it to the enumeration.

```
lemma (in enums) enum_append:
  assumes A1: A ∈ FinPow(X) and A2: b ∈ X-A and
  A3: ∀a∈A. a≤b
  shows  σ(A ∪ {b}) = σ(A)↩ b
proof -
  let f = σ(A) ∪ {⟨|A|,b⟩}
  from A1 have |A| ∈ nat using card_fin_is_nat
    by simp
  from A1 A2 have A ∪ {b} ∈ FinPow(X)
    using singleton_in_finpow union_finpow by simp
```

```
  moreover from this have |A ∪ {b}| ∈ nat
    using card_fin_is_nat by simp
  moreover have f ∈ ord_iso(|A ∪ {b}| , Le, A ∪ {b} ,r)
  proof -
    from A1 A2 have
      σ(A) ∈ ord_iso(|A|,Le, A,r) and
      |A| ∉ |A| and b ∉ A
      using enum_props  mem_not_refl by auto
    moreover from ‹|A| ∈ nat› have
      ∀k ∈ |A|. ⟨k, |A|⟩ ∈ Le
      using elem_nat_is_nat by blast
    moreover from A3 have ∀a∈A. ⟨a,b⟩ ∈ r by simp
    moreover have antisym(Le) and antisym(r)
      using linord NatOrder_ZF_1_L2 IsLinOrder_def by auto
    moreover
    from  A2 ‹|A| ∈ nat› have
      ⟨|A|,|A|⟩ ∈ Le and  ⟨b,b⟩ ∈ r
      using linord NatOrder_ZF_1_L2 IsLinOrder_def
 total_is_refl refl_def by auto
    hence ⟨|A|,|A|⟩ ∈ Le ⟷ ⟨b,b⟩ ∈ r by simp
    ultimately have f ∈ ord_iso(|A| ∪ {|A|} , Le, A ∪ {b} ,r)
      by (rule ord_iso_extend)
    with A1 A2 show f ∈ ord_iso(|A ∪ {b}| , Le, A ∪ {b} ,r)
      using card_fin_add_one by simp
  qed
  ultimately have f = σ(A ∪ {b})
    using ord_iso_enum by simp
  moreover have σ(A)↩ b = f
  proof -
    have σ(A)↩ b = σ(A) ∪ {⟨domain(σ(A)),b⟩}
      using Append_def by simp
    moreover from A1 have domain(σ(A)) = |A|
      using enum_props func1_1_L1 by blast
    ultimately show σ(A)↩ b = f by simp
  qed
  ultimately show σ(A ∪ {b}) = σ(A)↩ b by simp
qed
```

What is the enumeration of a singleton?

```
lemma (in enums) enum_singleton:
  assumes A1: x∈X shows σ({x}): 1 → X and σ({x})(0) = x
  proof -
    from A1 have
      0 ∈ FinPow(X) and x ∈ (X - 0) and ∀a∈0. a≤x
      using empty_in_finpow by auto
    then have σ(0 ∪ {x}) = σ(0)↩ x by (rule enum_append)
    with A1 show σ({x}): 1 → X and σ({x})(0) = x
      using empty_enum empty_append1 by auto
qed
```

end

24 Folding in ZF

theory `Fold_ZF` **imports** `InductiveSeq_ZF`

begin

Suppose we have a binary operation $P : X \times X \to X$ written multiplicatively as $P\langle x, y\rangle = x \cdot y$. In informal mathematics we can take a sequence $\{x_k\}_{k \in 0..n}$ of elements of X and consider the product $x_0 \cdot x_1 \cdot .. \cdot x_n$. To do the same thing in formalized mathematics we have to define precisely what is meant by that "$\cdot .. \cdot$". The definitition we want to use is based on the notion of sequence defined by induction discussed in `InductiveSeq_ZF`. We don't really want to derive the terminology for this from the word "product" as that would tie it conceptually to the multiplicative notation. This would be awkward when we want to reuse the same notions to talk about sums like $x_0 + x_1 + .. + x_n$. In functional programming there is something called "fold". Namely for a function f, initial point a and list $[b, c, d]$ the expression `fold(f, a, [b,c,d])` is defined to be `f(f(f(a,b),c),d)` (in Haskell something like this is called `foldl`). If we write f in multiplicative notation we get $a \cdot b \cdot c \cdot d$, so this is exactly what we need. The notion of folds in functional programming is actually much more general that what we need here (not that I know anything about that). In this theory file we just make a slight generalization and talk about folding a list with a binary operation $f : X \times Y \to X$ with X not necessarily the same as Y.

24.1 Folding in ZF

Suppose we have a binary operation $f : X \times Y \to X$. Then every $y \in Y$ defines a transformation of X defined by $T_y(x) = f\langle x, y\rangle$. In IsarMathLib such transformation is called as `Fix2ndVar(f,y)`. Using this notion, given a function $f : X \times Y \to X$ and a sequence $y = \{y_k\}_{k \in N}$ of elements of X we can get a sequence of transformations of X. This is defined in `Seq2TransSeq` below. Then we use that sequence of tranformations to define the sequence of partial folds (called `FoldSeq`) by means of `InductiveSeqVarFN` (defined in `InductiveSeq_ZF` theory) which implements the inductive sequence determined by a starting point and a sequence of transformations. Finally, we define the fold of a sequence as the last element of the sequence of the partial folds.

Definition that specifies how to convert a sequence a of elements of Y into a sequence of transformations of X, given a binary operation $f : X \times Y \to X$.

definition
```
Seq2TrSeq(f,a) ≡ {⟨k,Fix2ndVar(f,a(k))⟩. k ∈ domain(a)}
```

Definition of a sequence of partial folds.

definition
```
FoldSeq(f,x,a) ≡
InductiveSeqVarFN(x,fstdom(f),Seq2TrSeq(f,a),domain(a))
```

Definition of a fold.

definition
```
Fold(f,x,a) ≡ Last(FoldSeq(f,x,a))
```

If X is a set with a binary operation $f : X \times Y \to X$ then `Seq2TransSeqN(f,a)` converts a sequence a of elements of Y into the sequence of corresponding transformations of X.

```
lemma seq2trans_seq_props:
  assumes A1: n ∈ nat and A2: f : X×Y → X and A3: a:n→Y and
  A4: T = Seq2TrSeq(f,a)
  shows
  T : n → (X→X) and
  ∀k∈n. ∀x∈X. (T(k))(x) = f⟨x,a(k)⟩
proof -
  from ‹a:n→Y› have D: domain(a) = n using func1_1_L1 by simp
  with A2 A3 A4 show T : n → (X→X)
    using apply_funtype fix_2nd_var_fun ZF_fun_from_total Seq2TrSeq_def
    by simp
  with A4 D have I: ∀k ∈ n. T(k) = Fix2ndVar(f,a(k))
    using Seq2TrSeq_def ZF_fun_from_tot_val0 by simp
  { fix k fix x assume A5: k∈n  x∈X
    with A1 A3 have a(k) ∈ Y using apply_funtype
      by auto
    with A2 A5 I have (T(k))(x) = f⟨x,a(k)⟩
      using fix_var_val by simp
  } thus ∀k∈n. ∀x∈X. (T(k))(x) = f⟨x,a(k)⟩
    by simp
qed
```

Basic properties of the sequence of partial folds of a sequence $a = \{y_k\}_{k \in \{0,..,n\}}$.

```
theorem fold_seq_props:
  assumes A1: n ∈ nat and A2: f : X×Y → X and
  A3: y:n→Y and A4: x∈X and A5: Y≠0 and
  A6: F = FoldSeq(f,x,y)
  shows
  F: succ(n) → X
  F(0) = x and
  ∀k∈n. F(succ(k)) = f⟨F(k), y(k)⟩
proof -
  let T = Seq2TrSeq(f,y)
```

```
  from A1 A3 have D: domain(y) = n
    using func1_1_L1 by simp
  from ‹f : X×Y → X›  ‹Y≠0› have I: fstdom(f) = X
    using fstdomdef by simp
  with A1 A2 A3 A4 A6 D show
    II: F: succ(n) → X  and F(0) = x
    using seq2trans_seq_props FoldSeq_def fin_indseq_var_f_props
    by auto
  from A1 A2 A3 A4 A6 I D have ∀k∈n. F(succ(k)) = T(k)(F(k))
    using seq2trans_seq_props FoldSeq_def fin_indseq_var_f_props
    by simp
  moreover
  { fix k assume A5: k∈n hence k ∈ succ(n) by auto
    with A1 A2 A3 II A5 have (T(k))(F(k)) = f⟨F(k),y(k)⟩
      using apply_funtype seq2trans_seq_props by simp }
  ultimately show ∀k∈n. F(succ(k)) = f⟨F(k), y(k)⟩
    by simp
qed
```

A consistency condition: if we make the list shorter, then we get a shorter sequence of partial folds with the same values as in the original sequence. This can be proven as a special case of `fin_indseq_var_f_restrict` but a proof using `fold_seq_props` and induction turns out to be shorter.

```
lemma foldseq_restrict: assumes
  n ∈ nat   k ∈ succ(n) and
  i ∈ nat  f : X×Y → X  a : n → Y  b : i → Y and
  n ⊆ i   ∀j ∈ n. b(j) = a(j)   x ∈ X   Y ≠ 0
  shows FoldSeq(f,x,b)(k) = FoldSeq(f,x,a)(k)
proof -
  let P = FoldSeq(f,x,a)
  let Q = FoldSeq(f,x,b)
  from assms have
    n ∈ nat   k ∈ succ(n)
    Q(0) = P(0) and
    ∀j ∈ n. Q(j) = P(j) ⟶ Q(succ(j)) = P(succ(j))
    using fold_seq_props by auto
  then show  Q(k) = P(k) by (rule fin_nat_ind)
qed
```

A special case of `foldseq_restrict` when the longer sequence is created from the shorter one by appending one element.

```
corollary fold_seq_append:
  assumes n ∈ nat   f : X×Y → X   a:n → Y and
  x∈X   k ∈ succ(n)   y∈Y
  shows FoldSeq(f,x,Append(a,y))(k) = FoldSeq(f,x,a)(k)
proof -
  let b = Append(a,y)
  from assms have b : succ(n) → Y  ∀j ∈ n. b(j) = a(j)
    using append_props by auto
```

```
  with assms show thesis using foldseq_restrict by blast
qed
```

What we really will be using is the notion of the fold of a sequence, which we define as the last element of (inductively defined) sequence of partial folds. The next theorem lists some properties of the product of the fold operation.

```
theorem fold_props:
  assumes A1: n ∈ nat and
  A2: f : X×Y → X  a:n → Y  x∈X  Y≠0
  shows
  Fold(f,x,a) =  FoldSeq(f,x,a)(n) and
  Fold(f,x,a) ∈ X
proof -
  from assms have  FoldSeq(f,x,a) : succ(n) → X
    using  fold_seq_props by simp
  with A1 show
    Fold(f,x,a) =  FoldSeq(f,x,a)(n) and Fold(f,x,a) ∈ X
    using last_seq_elem apply_funtype Fold_def by auto
qed
```

A corner case: what happens when we fold an empty list?

```
theorem fold_empty: assumes A1: f : X×Y → X and
  A2: a:0→Y  x∈X  Y≠0
  shows Fold(f,x,a) = x
proof -
  let F = FoldSeq(f,x,a)
  from assms have I:
    0 ∈ nat  f : X×Y → X  a:0→Y  x∈X  Y≠0
    by auto
  then have Fold(f,x,a) = F(0) by (rule fold_props)
  moreover
  from I have
    0 ∈ nat  f : X×Y → X  a:0→Y  x∈X  Y≠0 and
    F = FoldSeq(f,x,a) by auto
  then have F(0) = x by (rule fold_seq_props)
  ultimately show Fold(f,x,a) = x by simp
qed
```

The next theorem tells us what happens to the fold of a sequence when we add one more element to it.

```
theorem fold_append:
  assumes A1: n ∈ nat and  A2: f : X×Y → X and
  A3: a:n→Y and A4: x∈X and A5: y∈Y
  shows
  FoldSeq(f,x,Append(a,y))(n) = Fold(f,x,a) and
  Fold(f,x,Append(a,y)) = f⟨Fold(f,x,a), y⟩
proof -
  let b = Append(a,y)
```

```
  let P = FoldSeq(f,x,b)
  from A5 have I: Y ≠ 0 by auto
  with assms show thesis1: P(n) = Fold(f,x,a)
    using fold_seq_append fold_props by simp
  from assms I have II:
    succ(n) ∈ nat   f : X×Y → X
    b : succ(n) → Y   x∈X  Y ≠ 0
    P = FoldSeq(f,x,b)
    using append_props by auto
  then have
    ∀k ∈ succ(n). P(succ(k)) =  f⟨P(k), b(k)⟩
    by (rule fold_seq_props)
  with A3 A5 thesis1 have P(succ(n)) =  f⟨ Fold(f,x,a), y⟩
    using append_props by auto
  moreover
  from II have P : succ(succ(n)) → X
    by (rule fold_seq_props)
  then have Fold(f,x,b) = P(succ(n))
    using last_seq_elem Fold_def by simp
  ultimately show Fold(f,x,Append(a,y)) = f⟨Fold(f,x,a), y⟩
    by simp
qed
```

Another way of formulating information contained in `fold_append` is to start with a longer sequence $a : n+1 \to X$ and then detach the last element from it. This provides an identity between the fold of the longer sequence and the value of the folding function on the fold of the shorter sequence and the last element of the longer one.

```
lemma fold_detach_last:
  assumes n ∈ nat f : X×Y → X x∈X ∀k∈n #+ 1. q(k) ∈ Y
  shows Fold(f,x,{⟨k,q(k)⟩. k∈n #+ 1}) = f⟨Fold(f,x,{⟨k,q(k)⟩. k∈n}),
q(n)⟩
proof -
  let a = {⟨k,q(k)⟩. k∈n #+ 1}
  let b = {⟨k,q(k)⟩. k∈n}
  from assms have
    Fold(f,x,Append(b,q(n))) = f⟨Fold(f,x,b), q(n)⟩
    using ZF_fun_from_total fold_append(2) by simp_all
  moreover from assms(1,4) have a = Append(b,q(n))
    using set_list_append1(4) by simp
  ultimately show Fold(f,x,a) = f⟨Fold(f,x,b), q(n)⟩
    by simp
qed
```

The tail of the sequence of partial folds defined by the folding function f, starting point x and a sequence y is the same as the sequence of partial folds starting from $f(x, y(0))$.

```
lemma fold_seq_detach_first:
```

```
  assumes n ∈ nat f : X×Y → X y:succ(n)→Y x∈X
  shows FoldSeq(f,f⟨x,y(0)⟩,Tail(y)) = Tail(FoldSeq(f,x,y))
proof -
  let F = FoldSeq(f,x,y)
  let T = Tail(F)
  let S = Seq2TrSeq(f,Tail(y))
  from assms(1,3) have succ(n) ∈ nat 0 ∈ succ(n) y(0)∈Y
    using empty_in_every_succ apply_funtype by simp_all
  have n ∈ nat f⟨x,y(0)⟩ ∈ X S:n→(X→X)
    and T:succ(n) → X T(0) = f⟨x,y(0)⟩
    and ∀k∈n. T(succ(k)) = (S(k))(T(k))
  proof -
    from assms(1) show n ∈ nat by simp
    from assms ‹succ(n) ∈ nat› show T:succ(n) → X
      using fold_seq_props(1) tail_props(1) nelist_vals_nonempty by simp
    from assms(2,4) ‹y(0)∈Y› show f⟨x,y(0)⟩ ∈ X
      using apply_funtype by simp
    from assms(1,2,3) show S:n→(X→X)
      using tail_props(1) seq2trans_seq_props by simp
    from assms  have I: F:succ(succ(n)) → X
      using fold_seq_props(1) nelist_vals_nonempty by simp
    show T(0) = f⟨x,y(0)⟩
    proof -
      from ‹succ(n) ∈ nat› ‹0 ∈ succ(n)› I
        have T(0) = F(succ(0))
        using tail_props(2) by blast
      moreover from assms  ‹0 ∈ succ(n)›
        have F(succ(0)) = f⟨F(0), y(0)⟩
        using fold_seq_props(3) nelist_vals_nonempty by blast
      moreover from assms  have F(0) = x
        using fold_seq_props(2) nelist_vals_nonempty by blast
      ultimately show T(0) = f⟨x,y(0)⟩ by simp
    qed
    show ∀k∈n. T(succ(k)) = (S(k))(T(k))
    proof -
      { fix k assume k∈n
        with assms(1) have
          succ(k) ∈ succ(n) k∈succ(n) succ(k) ∈ succ(succ(n))
          using succ_ineq by auto
        with ‹succ(n) ∈ nat› I have T(succ(k)) = F(succ(succ(k)))
          using tail_props(2) by blast
        moreover from assms ‹succ(k) ∈ succ(n)›
          have F(succ(succ(k))) = f⟨F(succ(k)), y(succ(k))⟩
            using fold_seq_props(3) nelist_vals_nonempty by blast
        moreover from assms(1,3) ‹k∈n› have y(succ(k)) = (Tail(y))(k)
          using tail_props(2) by simp
        moreover from assms ‹k∈n› I ‹succ(k) ∈ succ(succ(n))›
          have f⟨F(succ(k)),(Tail(y))(k)⟩ = (S(k))(F(succ(k)))
            using tail_props(1) seq2trans_seq_props(2) apply_funtype
```

```
          by simp
        moreover from ‹succ(n) ∈ nat› I ‹k∈succ(n)›
          have T(k) = F(succ(k))
            using tail_props(2) by blast
        ultimately have T(succ(k)) = (S(k))(T(k)) by simp
      } thus thesis by simp
    qed
  qed
  then have T = InductiveSeqVarFN(f⟨x,y(0)⟩,X,S,n)
    by (rule is_fin_indseq_var_f)
  moreover have fstdom(f) = X and domain(Tail(y)) = n
  proof -
    from assms(2,3) show fstdom(f) = X
      using fstdomdef nelist_vals_nonempty by simp
    from assms(1,3) show domain(Tail(y)) = n
      using tail_props(1) func1_1_L1 by blast
  qed
  ultimately show thesis unfolding FoldSeq_def by simp
qed
```

Taking a fold of a sequence y with a function f with the starting point x is the same as the fold starting from $f\langle x, y(0)\rangle$ of the tail of y.

```
lemma fold_detach_first:
  assumes n ∈ nat f : X×Y → X y:succ(n)→Y x∈X
  shows Fold(f,x,y) = Fold(f,f⟨x,y(0)⟩,Tail(y))
proof -
  from assms have FoldSeq(f,x,y):succ(succ(n))→X
    using fold_seq_props(1) nelist_vals_nonempty by simp
  with assms show thesis
    using last_tail_last fold_seq_detach_first unfolding Fold_def
    by simp
qed

end
```

25 Partitions of sets

```
theory Partitions_ZF imports Finite_ZF FiniteSeq_ZF

begin
```

It is a common trick in proofs that we divide a set into non-overlapping subsets. The first case is when we split the set into two nonempty disjoint sets. Here this is modeled as an ordered pair of sets and the set of such divisions of set X is called `Bisections(X)`. The second variation on this theme is a set-valued function (aren't they all in ZF?) whose values are nonempty and mutually disjoint.

25.1 Bisections

This section is about dividing sets into two non-overlapping subsets.

The set of bisections of a given set A is a set of pairs of nonempty subsets of A that do not overlap and their union is equal to A.

```
definition
  Bisections(X) = {p ∈ Pow(X)×Pow(X).
  fst(p)≠0 ∧ snd(p)≠0 ∧ fst(p)∩snd(p) = 0 ∧ fst(p)∪snd(p) = X}
```

Properties of bisections.

```
lemma bisec_props: assumes ⟨A,B⟩ ∈ Bisections(X) shows
  A≠0  B≠0 A ⊆ X  B ⊆ X  A ∩ B = 0  A ∪ B = X  X ≠ 0
  using assms Bisections_def by auto
```

Kind of inverse of `bisec_props`: a pair of nonempty disjoint sets form a bisection of their union.

```
lemma is_bisec:
  assumes A≠0  B≠0  A ∩ B = 0
  shows ⟨A,B⟩ ∈ Bisections(A∪B) using assms Bisections_def
  by auto
```

Bisection of X is a pair of subsets of X.

```
lemma bisec_is_pair: assumes Q ∈ Bisections(X)
  shows Q = ⟨fst(Q), snd(Q)⟩
  using assms Bisections_def by auto
```

The set of bisections of the empty set is empty.

```
lemma bisec_empty: shows Bisections(0) = 0
  using Bisections_def by auto
```

The next lemma shows what can we say about bisections of a set with another element added.

```
lemma bisec_add_point:
  assumes A1: x ∉ X and A2: ⟨A,B⟩ ∈ Bisections(X ∪ {x})
  shows (A = {x} ∨ B = {x}) ∨ (⟨A - {x}, B - {x}⟩ ∈ Bisections(X))
  proof -
    { assume A ≠ {x} and B ≠ {x}
      with A2 have A - {x} ≠ 0 and B - {x} ≠ 0
 using singl_diff_empty Bisections_def
 by auto
      moreover have (A - {x}) ∪ (B - {x}) = X
      proof -
 have (A - {x}) ∪ (B - {x}) = (A ∪ B) - {x}
   by auto
 also from assms have (A ∪ B) - {x} = X
   using Bisections_def by auto
```

```
 finally show thesis by simp
      qed
      moreover from A2 have (A - {x}) ∩ (B - {x}) = 0
 using Bisections_def by auto
      ultimately have ⟨A - {x}, B - {x}⟩ ∈ Bisections(X)
 using Bisections_def by auto
    } thus thesis by auto
qed
```

A continuation of the lemma `bisec_add_point` that refines the case when the pair with removed point bisects the original set.

```
lemma bisec_add_point_case3:
  assumes A1: ⟨A,B⟩ ∈ Bisections(X ∪ {x})
  and A2: ⟨A - {x}, B - {x}⟩ ∈ Bisections(X)
  shows
  (⟨A, B - {x}⟩ ∈ Bisections(X) ∧ x ∈ B) ∨
  (⟨A - {x}, B⟩ ∈ Bisections(X) ∧ x ∈ A)
proof -
  from A1 have x ∈ A ∪ B
    using Bisections_def by auto
  hence x∈A ∨ x∈B by simp
  from A1 have A - {x} = A ∨ B - {x} = B
    using Bisections_def by auto
  moreover
  { assume A - {x} = A
    with A2 ‹x ∈ A ∪ B› have
      ⟨A, B - {x}⟩ ∈ Bisections(X) ∧ x ∈ B
      using singl_diff_eq by simp }
  moreover
  { assume B - {x} = B
    with A2 ‹x ∈ A ∪ B› have
      ⟨A - {x}, B⟩ ∈ Bisections(X) ∧ x ∈ A
      using singl_diff_eq by simp }
  ultimately show thesis by auto
qed
```

Another lemma about bisecting a set with an added point.

```
lemma point_set_bisec:
  assumes A1: x ∉ X and A2: ⟨{x}, A⟩ ∈ Bisections(X ∪ {x})
  shows A = X and X ≠ 0
proof -
  from A2 have A ⊆ X using Bisections_def by auto
  moreover
  { fix a assume a∈X
    with A2 have a ∈ {x} ∪ A using Bisections_def by simp
    with A1 ‹a∈X› have a ∈ A by auto }
  ultimately show A = X by auto
  with A2 show X ≠ 0 using Bisections_def by simp
qed
```

Yet another lemma about bisecting a set with an added point, very similar to `point_set_bisec` with almost the same proof.

```
lemma set_point_bisec:
  assumes A1: x ∉ X and A2: ⟨A, {x}⟩ ∈ Bisections(X ∪ {x})
  shows A = X and X ≠ 0
proof -
  from A2 have A ⊆ X using Bisections_def by auto
  moreover
  { fix a assume a∈X
    with A2 have a ∈ A ∪ {x} using Bisections_def by simp
    with A1 ‹a∈X› have a ∈ A by auto }
  ultimately show A = X by auto
  with A2 show X ≠ 0 using Bisections_def by simp
qed
```

If a pair of sets bisects a finite set, then both elements of the pair are finite.

```
lemma bisect_fin:
  assumes A1: A ∈ FinPow(X) and A2: Q ∈ Bisections(A)
  shows fst(Q) ∈ FinPow(X) and snd(Q) ∈ FinPow(X)
proof -
  from A2 have ⟨fst(Q), snd(Q)⟩ ∈ Bisections(A)
    using bisec_is_pair by simp
  then have fst(Q) ⊆ A and snd(Q) ⊆ A
    using bisec_props by auto
  with A1 show fst(Q) ∈ FinPow(X) and snd(Q) ∈ FinPow(X)
    using FinPow_def subset_Finite by auto
qed
```

25.2 Partitions

This sections covers the situation when we have an arbitrary number of sets we want to partition into.

We define a notion of a partition as a set valued function such that the values for different arguments are disjoint. The name is derived from the fact that such function "partitions" the union of its arguments. Please let me know if you have a better idea for a name for such notion. We would prefer to say "is a partition", but that reserves the letter "a" as a keyword(?) which causes problems.

```
definition
  Partition (_ {is partition} [90] 91) where
  P {is partition} ≡  ∀x ∈ domain(P).
  P(x) ≠ 0 ∧ (∀y ∈ domain(P). x≠y ⟶ P(x) ∩ P(y) = 0)
```

A fact about lists of mutually disjoint sets.

```
lemma list_partition: assumes A1: n ∈ nat and
  A2: a : succ(n) → X   a {is partition}
```

```
  shows (⋃i∈n. a(i)) ∩ a(n) = 0
proof -
  { assume (⋃i∈n. a(i)) ∩ a(n) ≠ 0
    then have ∃x. x ∈ (⋃i∈n. a(i)) ∩ a(n)
      by (rule nonempty_has_element)
    then obtain x where x ∈ (⋃i∈n. a(i)) and  I: x ∈ a(n)
      by auto
    then obtain i where i ∈ n and x ∈ a(i) by auto
    with A2 I have False
      using mem_imp_not_eq func1_1_L1 Partition_def
      by auto
  } thus thesis by auto
qed
```

We can turn every injection into a partition.

```
lemma inj_partition:
  assumes A1: b ∈ inj(X,Y)
  shows
  ∀x ∈ X. {⟨x, {b(x)}⟩. x ∈ X}(x) = {b(x)} and
  {⟨x, {b(x)}⟩. x ∈ X} {is partition}
proof -
  let p = {⟨x, {b(x)}⟩. x ∈ X}
  { fix x assume x ∈ X
    from A1 have b : X → Y using inj_def
      by simp
    with ‹x ∈ X› have {b(x)} ∈ Pow(Y)
       using apply_funtype by simp
  } hence ∀x ∈ X. {b(x)} ∈ Pow(Y) by simp
  then have p : X → Pow(Y) using ZF_fun_from_total
    by simp
  then have domain(p) = X using func1_1_L1
    by simp
  from ‹p : X → Pow(Y)› show I: ∀x ∈ X. p(x) = {b(x)}
    using ZF_fun_from_tot_val0 by simp
  { fix x assume x ∈ X
    with I have p(x) = {b(x)} by simp
    hence p(x) ≠ 0 by simp
    moreover
    { fix t assume t ∈ X and x ≠ t
      with A1 ‹x ∈ X› have b(x) ≠ b(t) using inj_def
 by auto
      with I ‹x∈X› ‹t ∈ X› have p(x) ∩ p(t) = 0
 by auto }
    ultimately have
      p(x) ≠ 0 ∧ (∀t ∈ X. x≠t ⟶ p(x) ∩ p(t) = 0)
      by simp
  } with ‹domain(p) = X› show {⟨x, {b(x)}⟩. x ∈ X} {is partition}
    using Partition_def by simp
qed
```

end

26 Quasigroups

theory Quasigroup_ZF **imports** func1

begin

A quasigroup is an algebraic structure that that one gets after adding (sort of) divsibility to magma. Quasigroups differ from groups in that they are not necessarily associative and they do not have to have the neutral element.

26.1 Definitions and notation

According to Wikipedia there are at least two approaches to defining a quasigroup. One defines a quasigroup as a set with a binary operation, and the other, from universal algebra, defines a quasigroup as having three primitive operations. We will use the first approach.

A quasigroup operation does not have to have the neutral element. The left division is defined as the only solution to the equation $a \cdot x = b$ (using multiplicative notation). The next definition specifies what does it mean that an operation A has a left division on a set G.

definition
```
  HasLeftDiv(G,A) ≡ ∀a∈G.∀b∈G.∃!x. (x∈G ∧ A⟨a,x⟩ = b)
```

An operation A has the right division if for all elements $a, b \in G$ the equation $x \cdot a = b$ has a unique solution.

definition
```
  HasRightDiv(G,A) ≡ ∀a∈G.∀b∈G.∃!x. (x∈G ∧ A⟨x,a⟩ = b)
```

An operation that has both left and right division is said to have the Latin square property.

definition
```
  HasLatinSquareProp (infix {has Latin square property on} 65) where
    A {has Latin square property on} G ≡ HasLeftDiv(G,A) ∧ HasRightDiv(G,A)
```

A quasigroup is a set with a binary operation that has the Latin square property.

definition
```
  IsAquasigroup(G,A) ≡ A:G×G→G ∧ A {has Latin square property on} G
```

The uniqueness of the left inverse allows us to define the left division as a function. The union expression as the value of the function extracts the only element of the set of solutions of the equation $x \cdot z = y$ for given $\langle x, y\rangle = p \in G \times G$ using the identity $\bigcup\{x\} = x$.

```
definition
  LeftDiv(G,A) ≡ {⟨p,⋃{z∈G. A⟨fst(p),z⟩ = snd(p)}⟩.p∈G×G}
```

Similarly the right division is defined as a function on $G \times G$.

```
definition
  RightDiv(G,A) ≡ {⟨p,⋃{z∈G. A⟨z,fst(p)⟩ = snd(p)}⟩.p∈G×G}
```

Left and right divisions are binary operations on G.

```
lemma lrdiv_binop: assumes IsAquasigroup(G,A) shows
  LeftDiv(G,A):G×G→G and  RightDiv(G,A):G×G→G
proof -
  { fix p assume p∈G×G
    with assms have
      ⋃{x∈G. A⟨fst(p),x⟩ = snd(p)} ∈ G and  ⋃{x∈G. A⟨x,fst(p)⟩ = snd(p)}
∈ G
      unfolding IsAquasigroup_def HasLatinSquareProp_def HasLeftDiv_def
HasRightDiv_def
      using ZF1_1_L9(2) by auto
  } then show  LeftDiv(G,A):G×G→G and RightDiv(G,A):G×G→G
    unfolding LeftDiv_def RightDiv_def using ZF_fun_from_total by auto
qed
```

We will use multiplicative notation for the quasigroup operation. The right and left division will be denoted a/b and $a\backslash b$, resp.

```
locale quasigroup0 =
  fixes G A
  assumes qgroupassum: IsAquasigroup(G,A)

  fixes qgroper (infixl · 70)
  defines qgroper_def[simp]: x·y ≡ A⟨x,y⟩

  fixes leftdiv (infixl \ 70)
  defines leftdiv_def[simp]: x\y ≡LeftDiv(G,A)⟨x,y⟩

  fixes rightdiv (infixl / 70)
  defines rightdiv_def[simp]:x/y ≡RightDiv(G,A)⟨y,x⟩
```

The quasigroup operation is closed on G.

```
lemma (in quasigroup0) qg_op_closed: assumes x∈G y∈G
  shows x·y ∈ G
  using qgroupassum assms IsAquasigroup_def apply_funtype by auto
```

A couple of properties of right and left division:

```
lemma (in quasigroup0) lrdiv_props: assumes x∈G y∈G
  shows
```

∃!z. z∈G ∧ z·x = y y/x ∈ G (y/x)·x = y **and**
  ∃!z. z∈G ∧ x·z = y x\y ∈ G x·(x\y) = y
proof -
  let z_r = ⋃{z∈G. z·x = y}
  from qgroupassum **have** I: RightDiv(G,A):G×G→G **using** lrdiv_binop(2)
by simp
  with assms **have** RightDiv(G,A)⟨x,y⟩ = z_r
    unfolding RightDiv_def **using** ZF_fun_from_tot_val **by** auto
  moreover
  from qgroupassum assms **show** ∃!z. z∈G ∧ z·x = y
    unfolding IsAquasigroup_def HasLatinSquareProp_def HasRightDiv_def
by simp
  then have z_r·x = y **by** (rule ZF1_1_L9)
  ultimately show (y/x)·x = y **by** simp
  let z_l = ⋃{z∈G. x·z = y}
  from qgroupassum **have** II: LeftDiv(G,A):G×G→G **using** lrdiv_binop(1)
by simp
  with assms **have** LeftDiv(G,A)⟨x,y⟩ = z_l
    unfolding LeftDiv_def **using** ZF_fun_from_tot_val **by** auto
  moreover
  from qgroupassum assms **show** ∃!z. z∈G ∧ x·z = y
    unfolding IsAquasigroup_def HasLatinSquareProp_def HasLeftDiv_def
by simp
  then have x·z_l = y **by** (rule ZF1_1_L9)
  ultimately show x·(x\y) = y **by** simp
  from assms I II **show** y/x ∈ G **and** x\y ∈ G **using** apply_funtype **by** auto
qed

We can cancel the left element on both sides of an equation.

lemma (**in** quasigroup0) qg_cancel_left:
  assumes x∈G y∈G z∈G **and** x·y = x·z
  shows y=z
  using qgroupassum assms qg_op_closed lrdiv_props(4) **by** blast

We can cancel the right element on both sides of an equation.

lemma (**in** quasigroup0) qg_cancel_right:
  assumes x∈G y∈G z∈G **and** y·x = z·x
  shows y=z
  using qgroupassum assms qg_op_closed lrdiv_props(1) **by** blast

Two additional identities for right and left division:

lemma (**in** quasigroup0) lrdiv_ident: **assumes** x∈G y∈G
  shows (y·x)/x = y **and** x\(x·y) = y
proof -
  from assms **have** (y·x)/x ∈ G **and** ((y·x)/x)·x = y·x
    using qg_op_closed lrdiv_props(2,3) **by** auto
  with assms **show** (y·x)/x = y **using** qg_cancel_right **by** simp
  from assms **have** x\(x·y) ∈ G **and** x·(x\(x·y)) = x·y
    using qg_op_closed lrdiv_props(5,6) **by** auto

```
  with assms show x\(x·y) = y using qg_cancel_left by simp
qed

end
```

27 Loops

```
theory Loop_ZF imports Quasigroup_ZF

begin
```

This theory specifies the definition and proves basic properites of loops. Loops are very similar to groups, the only property that is missing is associativity of the operation.

27.1 Definitions and notation

In this section we define the notions of identity element and left and right inverse.

A loop is a quasigroup with an identity elemen.

```
definition IsAloop(G,A) ≡ IsAquasigroup(G,A) ∧ (∃e∈G. ∀x∈G. A⟨e,x⟩ = x
∧ A⟨x,e⟩ = x)
```

The neutral element for a binary operation $A : G \times G \to G$ is defined as the only element e of G such that $A\langle x, e\rangle = x$ and $A\langle e, x\rangle = x$ for all $x \in G$. Note that although the loop definition guarantees the existence of (some) such element(s) at this point we do not know if this element is unique. We can define this notion h ere but it will become usable only after we prove uniqueness.

```
definition
 TheNeutralElement(G,f) ≡
  ( THE e. e∈G ∧ (∀ g∈G. f⟨e,g⟩ = g ∧ f⟨g,e⟩ = g))
```

We will reuse the notation defined in the `quasigroup0` locale, just adding the assumption about the existence of a neutral element and notation for it.

```
locale loop0 = quasigroup0 +
  assumes ex_ident: ∃e∈G. ∀x∈G. e·x = x ∧ x·e = x

  fixes neut (1)
  defines neut_def[simp]: 1 ≡ TheNeutralElement(G,A)
```

In the loop context the pair `(G,A)` forms a loop.

```
lemma (in loop0) is_loop: shows IsAloop(G,A)
  unfolding IsAloop_def using ex_ident qgroupassum by simp
```

If we know that a pair (G,A) forms a loop then the assumptions of the loop0 locale hold.

```
lemma loop_loop0_valid: assumes IsAloop(G,A) shows loop0(G,A)
  using assms unfolding IsAloop_def loop0_axioms_def quasigroup0_def loop0_def
  by auto
```

The neutral element is unique in the loop.

```
lemma (in loop0) neut_uniq_loop: shows
  ∃!e. e∈G ∧ (∀x∈G. e·x = x ∧ x·e = x)
proof
  from ex_ident show ∃e. e ∈ G ∧ (∀x∈G. e · x = x ∧ x · e = x) by auto
next
  fix e y
  assume e ∈ G ∧ (∀x∈G. e · x = x ∧ x · e = x)  y ∈ G ∧ (∀x∈G. y · x
= x ∧ x · y = x)
  then have e·y =y and e·y = e by auto
  thus e=y by simp
qed
```

The neutral element as defined in the loop0 locale is indeed neutral.

```
lemma (in loop0) neut_props_loop: shows 1∈G and ∀x∈G. 1·x =x ∧ x·1
= x
proof -
  let n = THE e. e∈G ∧ (∀x∈G. e·x = x ∧ x·e = x)
  have 1 = TheNeutralElement(G,A) by simp
  then have 1 = n unfolding TheNeutralElement_def by simp
  moreover have n∈G ∧ (∀x∈G. n·x = x ∧ x·n = x) using neut_uniq_loop
theI
    by simp
  ultimately show 1∈G and ∀x∈G. 1·x = x ∧ x·1 = x
    by auto
qed
```

Every element of a loop has unique left and right inverse (which need not be the same). Here we define the left inverse as a function on G.

```
definition
  LeftInv(G,A) ≡ {⟨x,⋃{y∈G. A⟨y,x⟩ = TheNeutralElement(G,A)}⟩. x∈G}
```

Definition of the right inverse as a function on G:

```
definition
  RightInv(G,A) ≡ {⟨x,⋃{y∈G. A⟨x,y⟩ = TheNeutralElement(G,A)}⟩. x∈G}
```

In a loop G right and left inverses are functions on G.

```
lemma (in loop0) lr_inv_fun: shows LeftInv(G,A):G→G RightInv(G,A):G→G
  unfolding LeftInv_def RightInv_def
  using neut_props_loop lrdiv_props(1,4) ZF1_1_L9 ZF_fun_from_total
  by auto
```

Right and left inverses have desired properties.

```
lemma (in loop0) lr_inv_props: assumes x∈G
  shows
    LeftInv(G,A)(x) ∈ G (LeftInv(G,A)(x))·x = 1
    RightInv(G,A)(x) ∈ G x·(RightInv(G,A)(x)) = 1
proof -
  from assms show LeftInv(G,A)(x) ∈ G and RightInv(G,A)(x) ∈ G
    using lr_inv_fun apply_funtype by auto
  from assms have ∃!y. y∈G ∧ y·x = 1
    using neut_props_loop(1) lrdiv_props(1) by simp
  then have (⋃{y∈G.  y·x = 1})·x = 1
    by (rule ZF1_1_L9)
  with assms show (LeftInv(G,A)(x))·x = 1
    using lr_inv_fun(1) ZF_fun_from_tot_val unfolding LeftInv_def by simp
  from assms have ∃!y. y∈G ∧ x·y = 1
    using neut_props_loop(1) lrdiv_props(4) by simp
  then have x·(⋃{y∈G.  x·y = 1}) = 1
    by (rule ZF1_1_L9)
  with assms show x·(RightInv(G,A)(x)) = 1
    using lr_inv_fun(2) ZF_fun_from_tot_val unfolding RightInv_def by
simp
qed

end
```

28 Ordered loops

```
theory OrderedLoop_ZF imports Loop_ZF Order_ZF

begin
```

This theory file is about properties of loops (the algebraic structures introduced in IsarMathLib in the Loop_ZF theory) with an additional order relation that is in a way compatible with the loop's binary operation. The oldest reference I have found on the subject is [6].

28.1 Definition and notation

An ordered loop (G, A) is a loop with a partial order relation r that is "translation invariant" with respect to the loop operation A.

A triple (G, A, r) is an ordered loop if (G, A) is a loop and r is a relation on G (i.e. a subset of $G \times G$ with is a partial order and for all elements $x, y, z \in G$ the condition $\langle x, y\rangle \in r$ is equivalent to both $\langle A\langle x, z\rangle, A\langle x, z\rangle\rangle \in r$ and $\langle A\langle z, x\rangle, A\langle z, x\rangle\rangle \in r$. This looks a bit awkward in the basic set theory notation, but using the additive notation for the group operation and $x \leq y$

to instead of $\langle x, y\rangle \in r$ this just means that $x \leq y$ if and only if $x+z \leq y+z$ and $x \leq y$ if and only if $z + x \leq z + y$.

definition
```
  IsAnOrdLoop(L,A,r) ≡
  IsAloop(L,A) ∧ r⊆L×L ∧ IsPartOrder(L,r) ∧ (∀x∈L. ∀y∈L. ∀z∈L.
  ((⟨x,y⟩ ∈ r ⟷ ⟨A⟨ x,z⟩,A⟨y,z⟩⟩ ∈ r) ∧ (⟨x,y⟩ ∈ r ⟷ ⟨A⟨z,x⟩,A⟨z,y⟩⟩
∈ r )))
```

We define the set of nonnegative elements in the obvious way as $L^+ = \{x \in L : 0 \leq x\}$.

definition
```
  Nonnegative(L,A,r) ≡ {x∈L. ⟨ TheNeutralElement(L,A),x⟩ ∈ r}
```

The `PositiveSet(L,A,r)` is a set similar to `Nonnegative(L,A,r)`, but without the neutral element.

definition
```
  PositiveSet(L,A,r) ≡
  {x∈L. ⟨ TheNeutralElement(L,A),x⟩ ∈ r ∧ TheNeutralElement(L,A)≠ x}
```

We will use the additive notation for ordered loops.

locale `loop1 =`
```
  fixes L and A and r

  assumes ordLoopAssum: IsAnOrdLoop(L,A,r)

  fixes neut (0)
  defines neut_def[simp]: 0 ≡ TheNeutralElement(L,A)

  fixes looper (infixl + 69)
  defines looper_def[simp]: x + y ≡ A⟨ x,y⟩

  fixes lesseq (infix ≤ 68)
  defines lesseq_def [simp]: x ≤ y ≡ ⟨x,y⟩ ∈ r

  fixes sless (infix < 68)
  defines sless_def[simp]: x < y ≡ x≤y ∧ x≠y

  fixes nonnegative (L⁺)
  defines nonnegative_def [simp]: L⁺ ≡ Nonnegative(L,A,r)

  fixes positive (L₊)
  defines positive_def[simp]: L₊ ≡ PositiveSet(L,A,r)

  fixes leftdiv (- _ + _)
  defines leftdiv_def[simp]: -x+y ≡ LeftDiv(L,A)⟨x,y⟩

  fixes rightdiv (infixl - 69)
  defines rightdiv_def[simp]:x-y ≡ RightDiv(L,A)⟨y,x⟩
```

Theorems proven in the loop0 locale are valid in the loop1 locale

```
sublocale loop1 < loop0 L A looper
  using ordLoopAssum loop_loop0_valid unfolding IsAnOrdLoop_def by auto
```

The notation $-x+y$ and $x-y$ denotes left and right division, resp. These two operations are closed in a loop, see lemma lrdiv_binop in the Quasigroup_ZF theory. The next lemma reiterates that fact using the notation of the loop1 context.

```
lemma (in loop1) left_right_sub_closed: assumes x∈L y∈L
  shows (-x+y) ∈ L and x-y ∈ L
proof -
  from qgroupassum have LeftDiv(L,A):L×L →L and  RightDiv(L,A):L×L
→L
    using lrdiv_binop by simp_all
  with assms show (-x+y) ∈ L and x-y ∈ L
    using apply_funtype by simp_all
qed
```

In this context $x \leq y$ implies that both x and y belong to L.

```
lemma (in loop1) lsq_members: assumes x≤y shows x∈L and y∈L
  using ordLoopAssum assms IsAnOrdLoop_def by auto
```

In this context $x < y$ implies that both x and y belong to L.

```
lemma (in loop1) less_members: assumes x<y shows x∈L and y∈L
  using ordLoopAssum assms IsAnOrdLoop_def by auto
```

In an ordered loop the order is translation invariant.

```
lemma (in loop1) ord_trans_inv: assumes x≤y z∈L
  shows x+z ≤ y+z and z+x ≤ z+y
proof -
  from ordLoopAssum assms have
    (⟨x,y⟩ ∈ r ⟷ ⟨A⟨ x,z⟩,A⟨y,z⟩⟩ ∈ r) ∧ (⟨x,y⟩ ∈ r ⟷ ⟨A⟨z,x⟩,A⟨z,y⟩⟩
∈ r )
    using lsq_members unfolding IsAnOrdLoop_def by blast
  with assms(1) show x+z ≤ y+z and z+x ≤ z+y by auto
qed
```

In an ordered loop the strict order is translation invariant.

```
lemma (in loop1) strict_ord_trans_inv: assumes x<y z∈L
  shows x+z < y+z and z+x < z+y
proof -
  from assms have x+z ≤ y+z and z+x ≤ z+y
    using ord_trans_inv by auto
  moreover have x+z ≠ y+z and z+x ≠ z+y
  proof -
    { assume x+z = y+z
      with assms have x=y using less_members qg_cancel_right by blast
```

```
      with assms(1) have False by simp
    } thus x+z ≠ y+z by auto
    { assume z+x = z+y
      with assms have x=y using less_members qg_cancel_left by blast
      with assms(1) have False by simp
    } thus z+x ≠ z+y by auto
  qed
  ultimately show x+z < y+z and z+x < z+y
    by auto
qed
```

We can cancel an element from both sides of an inequality on the right side.

```
lemma (in loop1) ineq_cancel_right: assumes x∈L y∈L z∈L and x+z ≤
y+z
  shows  x≤y
proof -
  from ordLoopAssum assms(1,2,3) have ⟨x,y⟩ ∈ r ⟷ ⟨A⟨ x,z⟩,A⟨y,z⟩⟩ ∈
r
    unfolding IsAnOrdLoop_def by blast
  with assms(4) show x≤y by simp
qed
```

We can cancel an element from both sides of a strict inequality on the right side.

```
lemma (in loop1) strict_ineq_cancel_right: assumes x∈L y∈L z∈L and
x+z < y+z
  shows  x<y
  using assms ineq_cancel_right by auto
```

We can cancel an element from both sides of an inequality on the left side.

```
lemma (in loop1) ineq_cancel_left: assumes x∈L y∈L z∈L and z+x ≤ z+y

  shows  x≤y
proof -
  from ordLoopAssum assms(1,2,3) have ⟨x,y⟩ ∈ r ⟷ ⟨A⟨z,x⟩,A⟨z,y⟩⟩ ∈
r
    unfolding IsAnOrdLoop_def by blast
  with assms(4) show x≤y by simp
qed
```

We can cancel an element from both sides of a strict inequality on the left side.

```
lemma (in loop1) strict_ineq_cancel_left:
  assumes x∈L y∈L z∈L and z+x < z+y
  shows  x<y
  using assms ineq_cancel_left by auto
```

The definition of the nonnegative set in the notation used in the `loop1` locale:

lemma (**in** loop1) nonneg_definition:
  shows x ∈ L^+ ⟷ **0** ≤ x **using** ordLoopAssum IsAnOrdLoop_def Nonnegative_def
by auto

The nonnegative set is contained in the loop.

lemma (**in** loop1) nonneg_subset: **shows** L^+ ⊆ L
  using Nonnegative_def **by** auto

The positive set is contained in the loop.

lemma (**in** loop1) positive_subset: **shows** L_+ ⊆ L
  using PositiveSet_def **by** auto

The definition of the positive set in the notation used in the `loop1` locale:

lemma (**in** loop1) posset_definition:
  shows x ∈ L_+ ⟷ (**0**≤x ∧ x≠**0**)
  using ordLoopAssum IsAnOrdLoop_def PositiveSet_def **by** auto

Another form of the definition of the positive set in the notation used in the `loop1` locale:

lemma (**in** loop1) posset_definition1:
  shows x ∈ L_+ ⟷ **0**<x
  using ordLoopAssum IsAnOrdLoop_def PositiveSet_def **by** auto

The order in an ordered loop is antisymmeric.

lemma (**in** loop1) loop_ord_antisym: **assumes** x≤y **and** y≤x
  shows x=y
proof -
  from ordLoopAssum assms **have** antisym(r) ⟨x,y⟩ ∈ r ⟨y,x⟩ ∈ r
    unfolding IsAnOrdLoop_def IsPartOrder_def **by** auto
  then show x=y **by** (rule Fol1_L4)
qed

The loop order is transitive.

lemma (**in** loop1) loop_ord_trans: **assumes** x≤y **and** y≤z **shows** x≤z
proof -
  from ordLoopAssum assms **have** trans(r) **and** ⟨x,y⟩ ∈ r ∧ ⟨y,z⟩ ∈ r
    unfolding IsAnOrdLoop_def IsPartOrder_def **by** auto
  then have ⟨x,z⟩ ∈ r **by** (rule Fol1_L3)
  thus thesis **by** simp
qed

The loop order is reflexive.

lemma (**in** loop1) loop_ord_refl: **assumes** x∈L **shows** x≤x
  using assms ordLoopAssum **unfolding** IsAnOrdLoop_def IsPartOrder_def refl_def

  by simp

The neutral element is nonnegative.

```
lemma (in loop1) loop_zero_nonneg: shows 0∈L+
  using neut_props_loop(1) loop_ord_refl nonneg_definition
  by simp
```

A form of mixed transitivity for the strict order:

```
lemma (in loop1) loop_strict_ord_trans: assumes x≤y and y<z
  shows x<z
proof -
  from assms have x≤y and y≤z by auto
  then have x≤z by (rule loop_ord_trans)
  with assms show x<z using loop_ord_antisym by auto
qed
```

Another form of mixed transitivity for the strict order:

```
lemma (in loop1) loop_strict_ord_trans1: assumes x<y and y≤z
  shows x<z
proof -
  from assms have x≤y and y≤z by auto
  then have x≤z by (rule loop_ord_trans)
  with assms show x<z using loop_ord_antisym by auto
qed
```

Yet another form of mixed transitivity for the strict order:

```
lemma (in loop1) loop_strict_ord_trans2: assumes x<y and y<z
  shows x<z
proof -
  from assms have x≤y and y≤z by auto
  then have x≤z by (rule loop_ord_trans)
  with assms show x<z using loop_ord_antisym by auto
qed
```

We can move an element to the other side of an inequality. Well, not exactly, but our notation creates an illusion to that effect.

```
lemma (in loop1) lsq_other_side: assumes x≤y
  shows 0 ≤ -x+y  (-x+y) ∈ L+ 0 ≤ y-x (y-x) ∈ L+
proof -
  from assms have x∈L y∈L 0∈L (-x+y) ∈ L (y-x) ∈ L
    using lsq_members neut_props_loop(1) lrdiv_props(2,5) by auto
  then have x = x+0 and y = x+(-x+y) using neut_props_loop(2) lrdiv_props(6)
    by auto
  with assms have x+0 ≤ x+(-x+y) by simp
  with ‹x∈L› ‹0∈L› ‹(-x+y) ∈ L› show 0 ≤ -x+y using ineq_cancel_left
    by simp
  then show (-x+y) ∈ L+ using nonneg_definition by simp
  from ‹x∈L› ‹y∈L› have x = 0+x and y = (y-x)+x
    using neut_props_loop(2) lrdiv_props(3) by auto
  with assms have 0+x ≤ (y-x)+x by simp
  with ‹x∈L› ‹0∈L› ‹(y-x) ∈ L› show 0 ≤ y-x using ineq_cancel_right
```

```
    by simp
  then show (y-x) ∈ L+ using  nonneg_definition by simp
qed
```

We can move an element to the other side of a strict inequality.

```
lemma (in loop1) ls_other_side: assumes x<y
  shows 0 < -x+y  (-x+y) ∈ L+  0 < y-x (y-x) ∈ L+
proof -
  from assms have x∈L y∈L 0∈L (-x+y) ∈ L (y-x) ∈ L
    using lsq_members neut_props_loop(1) lrdiv_props(2,5) by auto
  then have x = x+0 and y = x+(-x+y) using neut_props_loop(2) lrdiv_props(6)
    by auto
  with assms have x+0 < x+(-x+y) by simp
  with ‹x∈L› ‹0∈L› ‹(-x+y) ∈ L› show 0 < -x+y using strict_ineq_cancel_left
    by simp
  then show (-x+y) ∈ L+ using posset_definition1 by simp
  from ‹x∈L› ‹y∈L› have x = 0+x and y = (y-x)+x
    using neut_props_loop(2) lrdiv_props(3) by auto
  with assms have 0+x < (y-x)+x by simp
  with ‹x∈L› ‹0∈L› ‹(y-x) ∈ L› show 0 < y-x using strict_ineq_cancel_right
    by simp
  then show (y-x) ∈ L+ using posset_definition1 by simp
qed
```

We can add sides of inequalities.

```
lemma (in loop1) add_ineq: assumes x≤y z≤t
  shows x+z ≤ y+t
proof -
  from assms have x+z ≤ y+z
    using lsq_members(1) ord_trans_inv(1) by simp
  with assms show thesis using lsq_members(2) ord_trans_inv(2) loop_ord_trans
    by simp
qed
```

We can add sides of strict inequalities. The proof uses a lemma that relies on the antisymmetry of the order relation.

```
lemma (in loop1) add_ineq_strict: assumes x<y z<t
  shows x+z < y+t
proof -
  from assms have x+z < y+z
    using less_members(1) strict_ord_trans_inv(1) by auto
  moreover from assms have y+z < y+t
    using less_members(2) strict_ord_trans_inv(2) by auto
  ultimately show thesis by (rule loop_strict_ord_trans2)
qed
```

We can add sides of inequalities one of which is strict.

```
lemma (in loop1) add_ineq_strict1: assumes x≤y z<t
```

```
  shows x+z < y+t and z+x < t+y
proof -
    from assms have x+z ≤ y+z
      using less_members(1) ord_trans_inv(1) by auto
    with assms show x+z < y+t
      using lsq_members(2) strict_ord_trans_inv(2) loop_strict_ord_trans
      by blast
    from assms have z+x < t+x
      using lsq_members(1) strict_ord_trans_inv(1) by simp
    with assms show z+x < t+y
      using less_members(2) ord_trans_inv(2) loop_strict_ord_trans1
      by blast
qed
```

Subtracting a positive element decreases the value, while adding a positive element increases the value.

```
lemma (in loop1) add_subtract_pos: assumes x∈L 0<y
  shows
    x-y < x (-y+x) < x x < x+y x < y+x
proof -
  from assms(2) have y∈L using less_members(2) by simp
  from assms(1) have x≤x
    using ordLoopAssum unfolding IsAnOrdLoop_def IsPartOrder_def refl_def
    by simp
  with assms(2) have x+0 < x+y
    using add_ineq_strict1(1) by simp
  with assms ‹y∈L› show x-y < x
    using neut_props_loop(2) lrdiv_props(3) lrdiv_props(2) strict_ineq_cancel_right
    by simp
  from assms(2) ‹x≤x› have 0+x < y+x
    using add_ineq_strict1(2) by simp
  with assms ‹y∈L› show (-y+x) < x
    using neut_props_loop(2) lrdiv_props(6) lrdiv_props(5) strict_ineq_cancel_left
    by simp
  from assms(1) ‹x+0 < x+y› ‹0+x < y+x›
    show x < x+y x < y+x
    using neut_props_loop(2) by simp_all
qed
```

In ordered loop if the order relation down-directs the set of positive elements L_+ then L_+ is a down-directed set (see `Order_ZF` for definitions of those related but different notions).

```
lemma (in loop1) down_directs_directed: assumes r {down-directs} L₊
  shows IsDownDirectedSet(L₊,r)
  using ordLoopAssum assms positive_subset down_directs_subset
  unfolding IsAnOrdLoop_def by auto

end
```

29 Semigroups

theory `Semigroup_ZF` **imports** `Partitions_ZF Fold_ZF Enumeration_ZF`

begin

It seems that the minimal setup needed to talk about a product of a sequence is a set with a binary operation. Such object is called "magma". However, interesting properties show up when the binary operation is associative and such alebraic structure is called a semigroup. In this theory file we define and study sequences of partial products of sequences of magma and semigroup elements.

29.1 Products of sequences of semigroup elements

Semigroup is a a magma in which the binary operation is associative. In this section we mostly study the products of sequences of elements of semigroup. The goal is to establish the fact that taking the product of a sequence is distributive with respect to concatenation of sequences, i.e for two sequences a, b of the semigroup elements we have $\prod(a \sqcup b) = (\prod a) \cdot (\prod b)$, where "$a \sqcup b$" is concatenation of a and b (a++b in Haskell notation). Less formally, we want to show that we can discard parantheses in expressions of the form $(a_0 \cdot a_1 \cdot .. \cdot a_n) \cdot (b_0 \cdot .. \cdot b_k)$.

First we define a notion similar to `Fold`, except that that the initial element of the fold is given by the first element of sequence. By analogy with Haskell fold we call that `Fold1`

definition
`Fold1(f,a) ≡ Fold(f,a(0),Tail(a))`

The definition of the `semigr0` context below introduces notation for writing about finite sequences and semigroup products. In the context we fix the carrier and denote it G. The binary operation on G is called f. All theorems proven in the context `semigr0` will implicitly assume that f is an associative operation on G. We will use multiplicative notation for the semigroup operation. The product of a sequence a is denoted $\prod a$. We will write $a \hookleftarrow x$ for the result of appending an element x to the finite sequence (list) a. This is a bit nonstandard, but I don't have a better idea for the "append" notation. Finally, $a \sqcup b$ will denote the concatenation of the lists a and b.

locale `semigr0 =`

fixes `G f`

assumes `assoc_assum: f {is associative on} G`

fixes `prod` (**infixl** `· 72`)

```
  defines prod_def [simp]: x · y ≡ f⟨x,y⟩

  fixes seqprod (∏ _ 71)
  defines seqprod_def [simp]: ∏ a ≡ Fold1(f,a)

  fixes append (infix ↩ 72)
  defines append_def [simp]: a ↩ x ≡ Append(a,x)

  fixes concat (infixl ⊔ 69)
  defines concat_def [simp]: a ⊔ b ≡ Concat(a,b)
```

The next lemma shows our assumption on the associativity of the semigroup operation in the notation defined in in the `semigr0` context.

```
lemma (in semigr0) semigr_assoc: assumes x ∈ G  y ∈ G  z ∈ G
  shows x·y·z = x·(y·z)
  using assms assoc_assum IsAssociative_def by simp
```

In the way we define associativity the assumption that f is associative on G also implies that it is a binary operation on X.

```
lemma (in semigr0) semigr_binop: shows f : G×G → G
  using assoc_assum IsAssociative_def by simp
```

Semigroup operation is closed.

```
lemma (in semigr0) semigr_closed:
  assumes a∈G  b∈G shows a·b ∈ G
  using assms semigr_binop apply_funtype by simp
```

Lemma `append_1elem` written in the notation used in the `semigr0` context.

```
lemma (in semigr0) append_1elem_nice:
  assumes n ∈ nat and a: n → X and b : 1 → X
  shows a ⊔ b = a ↩ b(0)
  using assms append_1elem by simp
```

Lemma `concat_init_last_elem` rewritten in the notation used in the `semigr0` context.

```
lemma (in semigr0) concat_init_last:
  assumes n ∈ nat  k ∈ nat and
  a: n → X  and b : succ(k) → X
  shows (a ⊔ Init(b)) ↩ b(k) = a ⊔ b
  using assms concat_init_last_elem by simp
```

The product of semigroup (actually, magma – we don't need associativity for this) elements is in the semigroup.

```
lemma (in semigr0) prod_type:
  assumes n ∈ nat and a : succ(n) → G
  shows (∏ a) ∈ G
proof -
```

```
  from assms have
    succ(n) ∈ nat  f : G×G → G  Tail(a) : n → G
    using semigr_binop tail_props by auto
  moreover from assms have a(0) ∈ G and G ≠ 0
    using empty_in_every_succ apply_funtype
    by auto
  ultimately show (∏ a) ∈ G using Fold1_def fold_props
    by simp
qed
```

What is the product of one element list?

```
lemma (in semigr0) prod_of_1elem: assumes A1: a: 1 → G
  shows (∏ a) = a(0)
proof -
  have f : G×G → G using semigr_binop by simp
  moreover from A1 have Tail(a) : 0 → G using tail_props
    by blast
  moreover from A1 have a(0) ∈ G and G ≠ 0
    using apply_funtype by auto
  ultimately show (∏ a) =  a(0) using fold_empty Fold1_def
    by simp
qed
```

What happens to the product of a list when we append an element to the list?

```
lemma (in semigr0) prod_append: assumes A1: n ∈ nat and
  A2: a : succ(n) → G and A3: x∈G
  shows (∏ a↩x) = (∏ a) · x
proof -
  from A1 A2 have I: Tail(a) : n → G  a(0) ∈ G
    using tail_props empty_in_every_succ apply_funtype
    by auto
  from assms have (∏ a↩x) = Fold(f,a(0),Tail(a)↩x)
    using head_of_append tail_append_commute Fold1_def
    by simp
  also from A1 A3 I have ... = (∏ a) · x
    using semigr_binop fold_append Fold1_def
    by simp
  finally show thesis by simp
qed
```

The main theorem of the section: taking the product of a sequence is distributive with respect to concatenation of sequences. The proof is by induction on the length of the second list.

```
theorem (in semigr0) prod_conc_distr:
  assumes A1: n ∈ nat  k ∈ nat and
  A2: a : succ(n) → G   b: succ(k) → G
  shows (∏ a) · (∏ b) = ∏ (a ⊔ b)
```

```
proof -
  from A1 have k ∈ nat by simp
  moreover have ∀b ∈ succ(0) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b)
  proof -
    { fix b assume A3: b : succ(0) → G
      with A1 A2 have
 succ(n) ∈ nat  a : succ(n) → G  b : 1 → G
 by auto
      then have a ⊔ b = a ↩ b(0) by (rule append_1elem_nice)
      with A1 A2 A3 have (∏ a) · (∏ b) = ∏ (a ⊔ b)
 using apply_funtype prod_append semigr_binop prod_of_1elem
 by simp
    } thus thesis by simp
  qed
  moreover have ∀j ∈ nat.
    (∀b ∈ succ(j) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b)) ⟶
    (∀b ∈ succ(succ(j)) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b))
  proof -
    { fix j assume A4: j ∈ nat and
      A5: (∀b ∈ succ(j) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b))
      { fix b assume A6: b : succ(succ(j)) → G
 let c = Init(b)
 from A4 A6 have  T: b(succ(j)) ∈ G and
   I: c : succ(j) → G and II: b = c↩b(succ(j))
   using apply_funtype init_props by auto
 from A1 A2 A4 A6 have
   succ(n) ∈ nat  succ(j) ∈ nat
   a : succ(n) → G  b : succ(succ(j)) → G
   by auto
 then have III: (a ⊔ c) ↩ b(succ(j)) = a ⊔ b
   by (rule concat_init_last)
 from A4 I T have (∏ c↩b(succ(j))) = (∏ c) · b(succ(j))
   by (rule prod_append)
 with II have
   (∏ a) · (∏ b) = (∏ a) · ((∏ c) · b(succ(j)))
   by simp
 moreover from A1 A2 A4 T I have
   (∏ a) ∈ G  (∏ c) ∈ G  b(succ(j)) ∈ G
   using prod_type by auto
 ultimately have
   (∏ a) · (∏ b) =  ((∏ a) · (∏ c)) · b(succ(j))
   using semigr_assoc by auto
 with A5 I have (∏ a) · (∏ b) = (∏ (a ⊔ c))·b(succ(j))
   by simp
 moreover
 from A1 A2 A4 I have
   T1: succ(n) ∈ nat  succ(j) ∈ nat and
   a : succ(n) → G   c : succ(j) → G
   by auto
```

```
 then have Concat(a,c): succ(n) #+ succ(j) → G
   by (rule concat_props)
 with A1 A4 T have
   succ(n #+ j) ∈ nat
   a ⊔ c : succ(succ(n #+j)) → G
   b(succ(j)) ∈ G
   using succ_plus by auto
 then have
   (∏ (a ⊔ c)↩b(succ(j))) = (∏ (a ⊔ c))·b(succ(j))
   by (rule prod_append)
 with III have (∏ (a ⊔ c))·b(succ(j)) =  ∏ (a ⊔ b)
   by simp
 ultimately have (∏ a) · (∏ b) = ∏ (a ⊔ b)
   by simp
      } hence (∀b ∈ succ(succ(j)) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b))
 by simp
    } thus thesis by blast
  qed
  ultimately have ∀b ∈ succ(k) → G. (∏ a) · (∏ b) = ∏ (a ⊔ b)
    by (rule ind_on_nat)
  with A2 show (∏ a) · (∏ b) = ∏ (a ⊔ b) by simp
qed
```

$a \cdot b \cdot (c \cdot d) = a \cdot (b \cdot c) \cdot d$ for semigrouop elements $a, b, c, d \in G$. The `Commutative semigroups` section below contains a couple of rearrangements that need commutativity of the semigroup operation, but this one uses only associativity, so it's here.

```
lemma (in semigr0) rearr4elem_assoc:
  assumes a∈G  b∈G  c∈G  d∈G
  shows a·b·(c·d) = a·(b·c)·d
proof -
  from assms have a·b·(c·d) = a·b·c·d using semigr_closed semigr_assoc
    by simp
  with assms(1,2,3) show thesis using semigr_assoc by simp
qed
```

29.2 Products over sets of indices

In this section we study the properties of expressions of the form $\prod_{i\in\Lambda} a_i = a_{i_0} \cdot a_{i_1} \cdot .. \cdot a_{i-1}$, i.e. what we denote as $\prod$`(Λ,a)`. Λ here is a finite subset of some set X and a is a function defined on X with values in the semigroup G.

Suppose $a : X \to G$ is an indexed family of elements of a semigroup G and $\Lambda = \{i_0, i_1, .., i_{n-1}\} \subseteq \mathbb{N}$ is a finite set of indices. We want to define $\prod_{i\in\Lambda} a_i = a_{i_0} \cdot a_{i_1} \cdot .. \cdot a_{i-1}$. To do that we use the notion of `Enumeration` defined in the `Enumeration_ZF` theory file that takes a set of indices and lists them in increasing order, thus converting it to list. Then we use the `Fold1` to multiply the resulting list. Recall that in Isabelle/ZF the capital letter "O" denotes the composition of two functions (or relations).

definition
```
  SetFold(f,a,Λ,r) = Fold1(f,a O Enumeration(Λ,r))
```

For a finite subset Λ of a linearly ordered set X we will write $\sigma(\Lambda)$ to denote the enumeration of the elements of Λ, i.e. the only order isomorphism $|\Lambda| \to \Lambda$, where $|\Lambda| \in \mathbb{N}$ is the number of elements of Λ. We also define notation for taking a product over a set of indices of some sequence of semigroup elements. The product of semigroup elements over some set $\Lambda \subseteq X$ of indices of a sequence $a : X \to G$ (i.e. $\prod_{i\in\Lambda} a_i$) is denoted `∏(Λ,a)`. In the `semigr1` context we assume that a is a function defined on some linearly ordered set X with values in the semigroup G.

```
locale semigr1 = semigr0 +

  fixes X r
  assumes linord: IsLinOrder(X,r)

  fixes a
  assumes a_is_fun: a : X → G

  fixes σ
  defines σ_def [simp]: σ(A) ≡ Enumeration(A,r)

  fixes setpr (∏)
  defines setpr_def [simp]: ∏(Λ,b) ≡ SetFold(f,b,Λ,r)
```

We can use the `enums` locale in the `semigr0` context.

```
lemma (in semigr1) enums_valid_in_semigr1: shows enums(X,r)
  using linord enums_def by simp
```

Definition of product over a set expressed in notation of the `semigr0` locale.

```
lemma (in semigr1) setproddef:
  shows ∏(Λ,a) = ∏ (a O σ(Λ))
  using SetFold_def by simp
```

A composition of enumeration of a nonempty finite subset of $\mathbb{N}$ with a sequence of elements of G is a nonempty list of elements of G. This implies that a product over set of a finite set of indices belongs to the (carrier of) semigroup.

```
lemma (in semigr1) setprod_type: assumes
  A1: Λ ∈ FinPow(X) and A2: Λ≠0
  shows
  ∃n ∈ nat . |Λ| = succ(n) ∧ a O σ(Λ) : succ(n) → G
  and ∏(Λ,a) ∈ G
proof -
```

```
  from assms obtain n where n ∈ nat and |Λ| = succ(n)
    using card_non_empty_succ by auto
  from A1 have σ(Λ) : |Λ| → Λ
    using enums_valid_in_semigr1 enums.enum_props
    by simp
  with A1 have a O σ(Λ): |Λ| → G
    using a_is_fun FinPow_def comp_fun_subset
    by simp
  with ‹n ∈ nat› and ‹|Λ| = succ(n)› show
    ∃n ∈ nat . |Λ| = succ(n) ∧ a O σ(Λ) : succ(n) → G
    by auto
  from  ‹n ∈ nat› ‹|Λ| = succ(n)› ‹a O σ(Λ): |Λ| → G›
  show ∏(Λ,a) ∈ G using prod_type setproddef
    by auto
qed
```

The `enum_append` lemma from the `Enemeration` theory specialized for natural numbers.

```
lemma (in semigr1) semigr1_enum_append:
  assumes Λ ∈ FinPow(X) and
  n ∈ X - Λ and ∀k∈Λ. ⟨k,n⟩ ∈ r
  shows σ(Λ ∪ {n}) = σ(Λ)↩ n
  using assms  FinPow_def enums_valid_in_semigr1
    enums.enum_append by simp
```

What is product over a singleton?

```
lemma (in semigr1) gen_prod_singleton:
  assumes A1: x ∈ X
  shows  ∏({x},a) = a(x)
proof -
  from A1 have σ({x}): 1 → X and  σ({x})(0) = x
    using enums_valid_in_semigr1 enums.enum_singleton
    by auto
  then show ∏({x},a) = a(x)
    using a_is_fun comp_fun setproddef prod_of_1elem
      comp_fun_apply by simp
qed
```

A generalization of `prod_append` to the products over sets of indices.

```
lemma (in semigr1) gen_prod_append:
  assumes
  A1: Λ ∈ FinPow(X) and A2: Λ ≠ 0 and
  A3: n ∈ X -  Λ and
  A4: ∀k∈Λ. ⟨k,n⟩ ∈ r
  shows ∏(Λ ∪ {n}, a) = (∏(Λ,a)) · a(n)
proof -
  have ∏(Λ ∪ {n}, a) =  ∏  (a O σ(Λ ∪ {n}))
    using setproddef by simp
  also from A1 A3 A4 have ... = ∏  (a O (σ(Λ)↩ n))
```

```
    using semigr1_enum_append by simp
  also have ... = ∏ ((a 0 σ(Λ))↩ a(n))
  proof -
    from A1 A3 have
      |Λ| ∈ nat and σ(Λ) : |Λ| → X and n ∈ X
      using card_fin_is_nat enums_valid_in_semigr1 enums.enum_fun
      by auto
    then show thesis using a_is_fun list_compose_append
      by simp
  qed
  also from assms have ... = (∏ (a 0 σ(Λ)))·a(n)
    using a_is_fun setprod_type apply_funtype prod_append
    by blast
  also have ... = (∏(Λ,a)) · a(n)
    using SetFold_def by simp
  finally show ∏(Λ ∪ {n}, a) = (∏(Λ,a)) · a(n)
    by simp
qed
```

Very similar to gen_prod_append: a relation between a product over a set of indices and the product over the set with the maximum removed.

```
lemma (in semigr1) gen_product_rem_point:
  assumes A1: A ∈ FinPow(X) and
  A2: n ∈ A and  A4: A - {n} ≠ 0 and
  A3: ∀k∈A. ⟨k, n⟩ ∈ r
  shows
  (∏(A - {n},a)) · a(n) = ∏(A, a)
proof -
  let Λ = A - {n}
  from A1 A2 have Λ ∈ FinPow(X) and n ∈ X -  Λ
    using fin_rem_point_fin FinPow_def by auto
  with A3 A4 have ∏(Λ ∪ {n}, a) = (∏(Λ,a)) · a(n)
    using a_is_fun gen_prod_append by blast
  with A2 show thesis using rem_add_eq by simp
qed
```

29.3 Commutative semigroups

Commutative semigroups are those whose operation is commutative, i.e. $a \cdot b = b \cdot a$. This implies that for any permutation $s : n \to n$ we have $\prod_{j=0}^{n} a_j = \prod_{j=0}^{n} a_{s(j)}$, or, closer to the notation we are using in the semigr0 context, $\prod a = \prod (a \circ s)$. Maybe one day we will be able to prove this, but for now the goal is to prove something simpler: that if the semigroup operation is commutative taking the product of a sequence is distributive with respect to the operation: $\prod_{j=0}^{n}(a_j \cdot b_j) = \left(\prod_{j=0}^{n} a_j)\right) \left(\prod_{j=0}^{n} b_j)\right)$. Many of the rearrangements (namely those that don't use the inverse) proven in the AbelianGroup_ZF theory hold in fact in semigroups. Some of them will be reproven in this section.

A rearrangement with 3 elements.

```
lemma (in semigr0) rearr3elems:
  assumes f {is commutative on} G and a∈G  b∈G  c∈G
  shows a·b·c = a·c·b
  using assms semigr_assoc IsCommutative_def by simp
```

A rearrangement of four elements.

```
lemma (in semigr0) rearr4elems:
  assumes A1: f {is commutative on} G and
  A2: a∈G  b∈G  c∈G  d∈G
  shows a·b·(c·d) = a·c·(b·d)
proof -
  from A2 have a·b·(c·d) = a·b·c·d
    using semigr_closed semigr_assoc by simp
  also have a·b·c·d =  a·c·(b·d)
  proof -
    from A1 A2 have a·b·c·d = c·(a·b)·d
      using IsCommutative_def semigr_closed
      by simp
    also from A2 have ... =  c·a·b·d
      using semigr_closed semigr_assoc
      by simp
    also from A1 A2 have ... = a·c·b·d
      using IsCommutative_def semigr_closed
      by simp
    also from A2 have ... = a·c·(b·d)
      using semigr_closed semigr_assoc
      by simp
    finally show  a·b·c·d =  a·c·(b·d) by simp
  qed
  finally show  a·b·(c·d) = a·c·(b·d)
    by simp
qed
```

We start with a version of `prod_append` that will shorten a bit the proof of the main theorem.

```
lemma (in semigr0) shorter_seq: assumes A1: k ∈ nat and
  A2: a ∈ succ(succ(k)) → G
  shows (∏ a) = (∏ Init(a)) · a(succ(k))
proof -
  let x = Init(a)
  from assms have
    a(succ(k)) ∈ G and x : succ(k) → G
    using apply_funtype init_props by auto
  with A1 have (∏ x↩a(succ(k))) = (∏ x) · a(succ(k))
    using prod_append by simp
  with assms show thesis using init_props
```

```
    by simp
qed
```

A lemma useful in the induction step of the main theorem.

```
lemma (in semigr0) prod_distr_ind_step:
  assumes A1: k ∈ nat and
  A2: a : succ(succ(k)) → G and
  A3: b : succ(succ(k)) → G and
  A4: c : succ(succ(k)) → G and
  A5: ∀j∈succ(succ(k)). c(j) = a(j) · b(j)
  shows
  Init(a) : succ(k) → G
  Init(b) : succ(k) → G
  Init(c) : succ(k) → G
  ∀j∈succ(k). Init(c)(j) = Init(a)(j) · Init(b)(j)
proof -
  from A1 A2 A3 A4 show
    Init(a) : succ(k) → G
    Init(b) : succ(k) → G
    Init(c) : succ(k) → G
    using init_props by auto
  from A1 have T: succ(k) ∈ nat by simp
  from T A2 have ∀j∈succ(k). Init(a)(j) = a(j)
    by (rule init_props)
  moreover from T A3 have ∀j∈succ(k). Init(b)(j) = b(j)
     by (rule init_props)
   moreover from T A4 have ∀j∈succ(k). Init(c)(j) = c(j)
     by (rule init_props)
   moreover from A5 have ∀j∈succ(k). c(j) = a(j) · b(j)
     by simp
   ultimately show ∀j∈succ(k). Init(c)(j) = Init(a)(j) · Init(b)(j)
     by simp
qed
```

For commutative operations taking the product of a sequence is distributive with respect to the operation. This version will probably not be used in applications, it is formulated in a way that is easier to prove by induction. For a more convenient formulation see `prod_comm_distrib`. The proof by induction on the length of the sequence.

```
theorem (in semigr0) prod_comm_distr:
  assumes A1: f {is commutative on} G and A2: n∈nat
  shows ∀ a b c.
  (a : succ(n)→G ∧ b : succ(n)→G ∧ c : succ(n)→G ∧
  (∀j∈succ(n). c(j) = a(j) · b(j))) ⟶
  (∏ c) = (∏ a) · (∏ b)
proof -
  note A2
  moreover have ∀ a b c.
    (a : succ(0)→G ∧ b : succ(0)→G ∧ c : succ(0)→G ∧
```

```
    (∀j∈succ(0). c(j) = a(j) · b(j))) ⟶
    (∏ c) = (∏ a) · (∏ b)
  proof -
    { fix a b c
      assume a : succ(0)→G ∧ b : succ(0)→G ∧ c : succ(0)→G ∧
 (∀j∈succ(0). c(j) = a(j) · b(j))
      then have
 I: a : 1→G  b : 1→G  c : 1→G and
 II: c(0) = a(0) · b(0) by auto
      from I have
 (∏ a) = a(0) and (∏ b) = b(0) and (∏ c) = c(0)
 using prod_of_1elem by auto
      with II have (∏ c) = (∏ a) · (∏ b) by simp
    } then show thesis using Fold1_def by simp
  qed
  moreover have ∀k ∈ nat.
    (∀ a b c.
    (a : succ(k)→G ∧ b : succ(k)→G ∧ c : succ(k)→G ∧
    (∀j∈succ(k). c(j) = a(j) · b(j))) ⟶
    (∏ c) = (∏ a) · (∏ b)) ⟶
    (∀ a b c.
    (a : succ(succ(k))→G ∧ b : succ(succ(k))→G ∧ c : succ(succ(k))→G
∧
    (∀j∈succ(succ(k)). c(j) = a(j) · b(j))) ⟶
    (∏ c) = (∏ a) · (∏ b))
  proof
    fix k assume k ∈ nat
    show (∀a b c.
      a ∈ succ(k) → G ∧
      b ∈ succ(k) → G ∧ c ∈ succ(k) → G ∧
      (∀j∈succ(k). c(j) = a(j) · b(j)) ⟶
      (∏ c) = (∏ a) · (∏ b)) ⟶
      (∀a b c.
      a ∈ succ(succ(k)) → G ∧
      b ∈ succ(succ(k)) → G ∧
      c ∈ succ(succ(k)) → G ∧
      (∀j∈succ(succ(k)). c(j) = a(j) · b(j)) ⟶
      (∏ c) = (∏ a) · (∏ b))
    proof
      assume A3: ∀a b c.
 a ∈ succ(k) → G ∧
 b ∈ succ(k) → G ∧ c ∈ succ(k) → G ∧
 (∀j∈succ(k). c(j) = a(j) · b(j)) ⟶
 (∏ c) = (∏ a) · (∏ b)
      show ∀a b c.
 a ∈ succ(succ(k)) → G ∧
 b ∈ succ(succ(k)) → G ∧
 c ∈ succ(succ(k)) → G ∧
 (∀j∈succ(succ(k)). c(j) = a(j) · b(j)) ⟶
```

```
 (∏ c) = (∏ a) · (∏ b)
      proof -
 { fix a b c
   assume
     a ∈ succ(succ(k)) → G ∧
     b ∈ succ(succ(k)) → G ∧
     c ∈ succ(succ(k)) → G ∧
     (∀j∈succ(succ(k)). c(j) = a(j) · b(j))
   with ‹k ∈ nat› have I:
     a : succ(succ(k)) → G
     b : succ(succ(k)) → G
     c : succ(succ(k)) → G
     and II: ∀j∈succ(succ(k)). c(j) = a(j) · b(j)
     by auto
   let x = Init(a)
          let y = Init(b)
          let z = Init(c)
   from ‹k ∈ nat› I have III:
     (∏ a) = (∏ x) · a(succ(k))
     (∏ b) = (∏ y) · b(succ(k)) and
     IV: (∏ c) = (∏ z) · c(succ(k))
     using  shorter_seq by auto
   moreover
   from  ‹k ∈ nat› I II have
     x : succ(k) → G
     y : succ(k) → G
     z : succ(k) → G and
     ∀j∈succ(k). z(j) = x(j) · y(j)
     using prod_distr_ind_step by auto
   with A3 II IV have
     (∏ c) = (∏ x)·(∏ y)·(a(succ(k)) · b(succ(k)))
     by simp
   moreover from A1 ‹k ∈ nat› I III have
     (∏ x)·(∏ y)·(a(succ(k)) · b(succ(k)))=
     (∏ a) · (∏ b)
     using init_props prod_type apply_funtype
       rearr4elems by simp
   ultimately have (∏ c) = (∏ a) · (∏ b)
     by simp
 } thus thesis by auto
      qed
    qed
  qed
  ultimately show thesis by (rule ind_on_nat)
qed
```

A reformulation of `prod_comm_distr` that is more convenient in applications.

```
theorem (in semigr0) prod_comm_distrib:
  assumes  f {is commutative on} G and n∈nat and
```

```
  a : succ(n)→G  b : succ(n)→G  c : succ(n)→G and
  ∀j∈succ(n). c(j) = a(j) · b(j)
  shows (∏ c) = (∏ a) · (∏ b)
  using assms prod_comm_distr by simp
```

A product of two products over disjoint sets of indices is the product over the union.

```
lemma (in semigr1) prod_bisect:
  assumes  A1: f {is commutative on} G  and A2: Λ ∈ FinPow(X)
  shows
  ∀P ∈  Bisections(Λ). ∏(Λ,a) = (∏(fst(P),a))·(∏(snd(P),a))
proof -
  have IsLinOrder(X,r) using linord by simp
  moreover have
    ∀P ∈  Bisections(0). ∏(0,a) = (∏(fst(P),a))·(∏(snd(P),a))
    using bisec_empty by simp
  moreover have ∀ A ∈ FinPow(X).
    ( ∀ n ∈ X - A.
    (∀P ∈  Bisections(A). ∏(A,a) = (∏(fst(P),a))·(∏(snd(P),a)))
    ∧ (∀k∈A. ⟨k,n⟩ ∈ r ) ⟶
    (∀Q ∈  Bisections(A ∪ {n}).
    ∏(A ∪ {n},a) = (∏(fst(Q),a))·(∏(snd(Q),a))))
  proof -
    { fix A assume A ∈ FinPow(X)
      fix n assume n ∈ X - A
      have ( ∀P ∈ Bisections(A).
 ∏(A,a) = (∏(fst(P),a))·(∏(snd(P),a)))
 ∧ (∀k∈A. ⟨k,n⟩ ∈ r )  ⟶
 (∀Q ∈  Bisections(A ∪ {n}).
 ∏(A ∪ {n},a) = (∏(fst(Q),a))·(∏(snd(Q),a)))
      proof -
 { assume I:
   ∀P ∈ Bisections(A). ∏(A,a) = (∏(fst(P),a))·(∏(snd(P),a))
   and II: ∀k∈A. ⟨k,n⟩ ∈ r
   have ∀Q ∈  Bisections(A ∪ {n}).
     ∏(A ∪ {n},a) = (∏(fst(Q),a))·(∏(snd(Q),a))
   proof -
     { fix Q assume Q ∈  Bisections(A ∪ {n})
       let Q0 = fst(Q)
       let Q1 = snd(Q)
       from <A ∈ FinPow(X)> <n ∈ X - A> have A ∪ {n} ∈ FinPow(X)
  using singleton_in_finpow union_finpow by auto
       with <Q ∈  Bisections(A ∪ {n})> have
  Q0 ∈ FinPow(X) Q0 ≠ 0 and Q1 ∈ FinPow(X) Q1 ≠ 0
  using bisect_fin bisec_is_pair Bisections_def by auto
       then have ∏(Q0,a) ∈ G and ∏(Q1,a) ∈ G
  using a_is_fun setprod_type by auto
       from <Q ∈ Bisections(A ∪ {n})> <A ∈ FinPow(X)> <n ∈ X-A>
       have refl(X,r)  Q0 ⊆ A ∪ {n}  Q1 ⊆ A ∪ {n}
```

```
A ⊆ X and n ∈ X
using linord IsLinOrder_def total_is_refl Bisections_def
FinPow_def by auto
     from ‹refl(X,r)› ‹Q0 ⊆ A ∪ {n}› ‹A ⊆ X› ‹n ∈ X› II
     have III: ∀k ∈ Q0. ⟨k, n⟩ ∈ r by (rule refl_add_point)
     from ‹refl(X,r)› ‹Q1 ⊆ A ∪ {n}› ‹A ⊆ X› ‹n ∈ X› II
     have IV: ∀k ∈ Q1. ⟨k, n⟩ ∈ r by (rule refl_add_point)
     from ‹n ∈ X - A› ‹Q ∈ Bisections(A ∪ {n})› have
Q0 = {n} ∨ Q1 = {n} ∨ ⟨Q0 - {n},Q1-{n}⟩ ∈ Bisections(A)
using bisec_is_pair bisec_add_point by simp
     moreover
     { assume Q1 = {n}
from ‹n ∈ X - A› have n ∉ A by auto
moreover
from ‹Q ∈ Bisections(A ∪ {n})›
have ⟨Q0,Q1 ⟩ ∈ Bisections(A ∪ {n})
  using bisec_is_pair by simp
with ‹Q1 = {n}› have ⟨Q0, {n}⟩ ∈ Bisections(A ∪ {n})
  by simp
ultimately have Q0 = A and A ≠ 0
  using set_point_bisec by auto
with ‹A ∈ FinPow(X)› ‹n ∈ X - A› II ‹Q1 = {n}›
have ∏(A ∪ {n},a) = (∏(Q0,a))·∏(Q1,a)
  using a_is_fun gen_prod_append gen_prod_singleton
  by simp }
     moreover
     { assume Q0 = {n}
from ‹n ∈ X - A› have n ∈ X by auto
then have {n} ∈ FinPow(X) and {n} ≠ 0
  using singleton_in_finpow by auto
from ‹n ∈ X - A› have n ∉ A by auto
moreover
from ‹Q ∈ Bisections(A ∪ {n})›
have ⟨Q0, Q1⟩ ∈ Bisections(A ∪ {n})
  using bisec_is_pair by simp
with ‹Q0 = {n}› have ⟨{n}, Q1⟩ ∈ Bisections(A ∪ {n})
  by simp
ultimately have Q1 = A and A ≠ 0 using point_set_bisec
  by auto
with A1 ‹A ∈ FinPow(X)› ‹n ∈ X - A› II
  ‹{n} ∈ FinPow(X)› ‹{n} ≠ 0› ‹Q0 = {n}›
have ∏(A ∪ {n},a) = (∏(Q0,a))·(∏(Q1,a))
  using a_is_fun gen_prod_append gen_prod_singleton
    setprod_type IsCommutative_def by auto }
     moreover
     { assume A4: ⟨Q0 - {n},Q1 - {n}⟩ ∈ Bisections(A)
with ‹A ∈ FinPow(X)› have
  Q0 - {n} ∈ FinPow(X) Q0 - {n} ≠ 0 and
  Q1 - {n} ∈ FinPow(X) Q1 - {n} ≠ 0
```

```
    using FinPow_def Bisections_def by auto
  with ‹n ∈ X - A› have
    ∏(Q₀ - {n},a) ∈ G  ∏(Q₁ - {n},a) ∈ G  and
    T: a(n) ∈ G
    using a_is_fun setprod_type apply_funtype by auto
  from ‹Q ∈ Bisections(A ∪ {n})› A4 have
    (⟨Q₀, Q₁ - {n}⟩ ∈ Bisections(A) ∧ n ∈ Q₁) ∨
    (⟨Q₀ - {n}, Q₁⟩ ∈ Bisections(A) ∧ n ∈ Q₀)
    using bisec_is_pair bisec_add_point_case3 by auto
  moreover
  { assume ⟨Q₀, Q₁ - {n}⟩ ∈ Bisections(A) and n ∈ Q₁
    then have A ≠ 0 using bisec_props by simp
    with A2 ‹A ∈ FinPow(X)› ‹n ∈ X - A› I II T IV
      ‹⟨Q₀, Q₁ - {n}⟩ ∈ Bisections(A)› ‹∏(Q₀,a) ∈ G›
      ‹∏(Q₁ - {n},a) ∈ G› ‹Q₁ ∈ FinPow(X)›
      ‹n ∈ Q₁› ‹Q₁ - {n} ≠ 0›
    have ∏(A ∪ {n},a) = (∏(Q₀,a))·(∏(Q₁,a))
      using gen_prod_append semigr_assoc gen_product_rem_point
      by simp }
  moreover
  { assume ⟨Q₀ - {n}, Q₁⟩ ∈ Bisections(A) and n ∈ Q₀
    then have A ≠ 0 using bisec_props by simp
    with A1 A2 ‹A ∈ FinPow(X)› ‹n ∈ X - A› I II III T
      ‹⟨Q₀ - {n}, Q₁⟩∈Bisections(A)› ‹∏(Q₀ - {n},a)∈G›
      ‹∏(Q₁,a) ∈ G› ‹Q₀ ∈ FinPow(X)› ‹n ∈ Q₀› ‹Q₀-{n}≠0›
    have ∏(A ∪ {n},a) = (∏(Q₀,a))·(∏(Q₁,a))
      using gen_prod_append rearr3elems gen_product_rem_point
        by simp }
  ultimately have
    ∏(A ∪ {n},a) = (∏(Q₀,a))·(∏(Q₁,a))
    by auto }
      ultimately have ∏(A ∪ {n},a) = (∏(Q₀,a))·(∏(Q₁,a))
  by auto
     } thus thesis by simp
   qed
 } thus thesis by simp
     qed
   } thus thesis by simp
  qed
  moreover note A2
  ultimately show thesis by (rule fin_ind_add_max)
qed
```

A better looking reformulation of prod_bisect.

```
theorem (in semigr1) prod_disjoint: assumes
  A1: f {is commutative on} G  and
  A2: A ∈ FinPow(X) A ≠ 0 and
  A3: B ∈ FinPow(X) B ≠ 0 and
  A4: A ∩ B = 0
```

```
  shows ∏(A∪B,a) = (∏(A,a))·(∏(B,a))
proof -
  from A2 A3 A4 have ⟨A,B⟩ ∈ Bisections(A∪B)
    using is_bisec by simp
  with A1 A2 A3 show thesis
    using a_is_fun union_finpow prod_bisect by simp
qed
```

A generalization of `prod_disjoint`.

```
lemma (in semigr1) prod_list_of_lists: assumes
  A1: f {is commutative on} G  and A2: n ∈ nat
  shows ∀M ∈ succ(n) → FinPow(X).
  M {is partition} ⟶
  (∏ {⟨i,∏(M(i),a)⟩. i ∈ succ(n)}) =
  (∏(⋃i ∈ succ(n). M(i),a))
proof -
  note A2
  moreover have ∀M ∈ succ(0) → FinPow(X).
    M {is partition} ⟶
    (∏ {⟨i,∏(M(i),a)⟩. i ∈ succ(0)}) = (∏(⋃i ∈ succ(0). M(i),a))
    using a_is_fun func1_1_L1 Partition_def apply_funtype setprod_type
      list_len1_singleton prod_of_1elem
    by simp
  moreover have ∀k ∈ nat.
    (∀M ∈ succ(k) → FinPow(X).
    M {is partition} ⟶
    (∏ {⟨i,∏(M(i),a)⟩. i ∈ succ(k)}) =
    (∏(⋃i ∈ succ(k). M(i),a))) ⟶
    (∀M ∈ succ(succ(k)) → FinPow(X).
    M {is partition} ⟶
    (∏ {⟨i,∏(M(i),a)⟩. i ∈ succ(succ(k))}) =
    (∏(⋃i ∈ succ(succ(k)). M(i),a)))
  proof -
    { fix k assume k ∈ nat
      assume A3: ∀M ∈ succ(k) → FinPow(X).
 M {is partition} ⟶
 (∏ {⟨i,∏(M(i),a)⟩. i ∈ succ(k)}) =
 (∏(⋃i ∈ succ(k). M(i),a))
      have (∀N ∈ succ(succ(k)) → FinPow(X).
 N {is partition} ⟶
 (∏ {⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))}) =
 (∏(⋃i ∈ succ(succ(k)). N(i),a)))
      proof -
 { fix N assume A4: N : succ(succ(k)) → FinPow(X)
   assume A5: N {is partition}
   with A4 have I: ∀i ∈ succ(succ(k)). N(i) ≠ 0
     using func1_1_L1 Partition_def by simp
   let b = {⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))}
   let c = {⟨i,∏(N(i),a)⟩. i ∈ succ(k)}
```

```
have II: ∀i ∈ succ(succ(k)). ∏(N(i),a) ∈ G
proof
  fix i assume i ∈ succ(succ(k))
  with A4 I have N(i) ∈ FinPow(X) and N(i) ≠ 0
    using apply_funtype by auto
  then show ∏(N(i),a) ∈ G using setprod_type
    by simp
qed
hence ∀i ∈ succ(k).  ∏(N(i),a) ∈ G by auto
then have c : succ(k) → G by (rule ZF_fun_from_total)
have b = {⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))}
  by simp
with II have b = Append(c,∏(N(succ(k)),a))
  by (rule set_list_append)
with  II  ‹k ∈ nat› ‹c : succ(k) → G›
have (∏ b) = (∏ c)·(∏(N(succ(k)),a))
  using prod_append by simp
also have
  ... =  (∏(⋃i ∈ succ(k). N(i),a))·(∏(N(succ(k)),a))
proof -
  let M = restrict(N,succ(k))
  have succ(k) ⊆ succ(succ(k)) by auto
  with ‹N : succ(succ(k)) → FinPow(X)›
  have M : succ(k) → FinPow(X) and
    III: ∀i ∈ succ(k). M(i) = N(i)
    using restrict_type2 restrict apply_funtype
    by auto
  with A5 ‹M : succ(k) → FinPow(X)›have M {is partition}
    using func1_1_L1 Partition_def by simp
  with A3 ‹M : succ(k) → FinPow(X)› have
    (∏ {⟨i,∏(M(i),a)⟩. i ∈ succ(k)}) =
    (∏(⋃i ∈ succ(k). M(i),a))
    by blast
  with III show thesis by simp
qed
also have ... = (∏(⋃i ∈ succ(succ(k)). N(i),a))
proof -
  let A = ⋃i ∈ succ(k). N(i)
  let B = N(succ(k))
  from A4 ‹k ∈ nat› have succ(k) ∈ nat and
    ∀i ∈ succ(k). N(i) ∈ FinPow(X)
    using apply_funtype by auto
  then have A ∈ FinPow(X) by (rule union_fin_list_fin)
  moreover from I have A ≠ 0 by auto
  moreover from A4 I have
    N(succ(k)) ∈ FinPow(X) and N(succ(k)) ≠ 0
    using apply_funtype by auto
  moreover from ‹succ(k) ∈ nat› A4 A5 have A ∩ B = 0
    by (rule list_partition)
```

```
      moreover note A1
      ultimately have ∏(A∪B,a) = (∏(A,a))·(∏(B,a))
        using prod_disjoint by simp
      moreover have A ∪ B = (⋃i ∈ succ(succ(k)). N(i))
        by auto
      ultimately show thesis by simp
    qed
    finally have (∏ {⟨i,∏(N(i),a)⟩. i ∈ succ(succ(k))}) =
      (∏(⋃i ∈ succ(succ(k)). N(i),a))
      by simp
    } thus thesis by auto
  qed
  } thus thesis by simp
  qed
  ultimately show thesis by (rule ind_on_nat)
qed
```

A more convenient reformulation of `prod_list_of_lists`.

```
theorem (in semigr1) prod_list_of_sets:
  assumes A1: f {is commutative on} G  and
  A2: n ∈ nat  n ≠ 0 and
  A3: M : n → FinPow(X)   M {is partition}
  shows
  (∏ {⟨i,∏(M(i),a)⟩. i ∈ n}) = (∏(⋃i ∈ n. M(i),a))
proof -
  from A2 obtain k where k ∈ nat and n = succ(k)
    using Nat_ZF_1_L3 by auto
  with A1 A3 show thesis using prod_list_of_lists
    by simp
qed
```

The definition of the product `∏(A,a) ≡ SetFold(f,a,A,r)` of a some (finite) set of semigroup elements requires that r is a linear order on the set of indices A. This is necessary so that we know in which order we are multiplying the elements. The product over A is defined so that we have $\prod_A a = \prod a \circ \sigma(A)$ where $\sigma : |A| \rightarrow A$ is the enumeration of A (the only order isomorphism between the number of elements in A and A), see lemma `setproddef`. However, if the operation is commutative, the order is irrelevant. The next theorem formalizes that fact stating that we can replace the enumeration $\sigma(A)$ by any bijection between $|A|$ and A. In a way this is a generalization of `setproddef`. The proof is based on application of `prod_list_of_sets` to the finite collection of singletons that comprise A.

```
theorem (in semigr1) prod_order_irr:
  assumes A1: f {is commutative on} G and
  A2: A ∈ FinPow(X) A ≠ 0 and
  A3: b ∈ bij(|A|,A)
  shows (∏ (a O b)) = ∏(A,a)
proof -
```

```
 let n = |A|
 let M = {⟨k, {b(k)}⟩. k ∈ n}
 have (∏ (a O b)) = (∏ {⟨i,∏(M(i),a)⟩. i ∈ n})
 proof -
   have ∀i ∈ n. ∏(M(i),a) = (a O b)(i)
   proof
     fix i assume i ∈ n
     with A2 A3 ‹i ∈ n› have b(i) ∈ X
using bij_def inj_def apply_funtype FinPow_def
by auto
     then have ∏({b(i)},a) = a(b(i))
using gen_prod_singleton by simp
     with A3 ‹i ∈ n› have ∏({b(i)},a) = (a O b)(i)
using bij_def inj_def comp_fun_apply by auto
     with ‹i ∈ n› A3 show ∏(M(i),a) = (a O b)(i)
using bij_def inj_partition by auto
   qed
   hence {⟨i,∏(M(i),a)⟩. i ∈ n} = {⟨i,(a O b)(i)⟩. i ∈ n}
     by simp
   moreover have {⟨i,(a O b)(i)⟩. i ∈ n} = a O b
   proof -
     from A3 have b : n → A using bij_def inj_def by simp
     moreover from A2 have A ⊆ X using FinPow_def by simp
     ultimately have b : n → X by (rule func1_1_L1B)
     then have a O b: n → G using a_is_fun comp_fun
by simp
     then show {⟨i,(a O b)(i)⟩. i ∈ n} = a O b
using fun_is_set_of_pairs by simp
   qed
   ultimately show thesis by simp
 qed
 also have ... = (∏(⋃i ∈ n. M(i),a))
 proof -
   note A1
   moreover from A2 have n ∈ nat and n ≠ 0
     using card_fin_is_nat card_non_empty_non_zero by auto
   moreover have M : n → FinPow(X) and M {is partition}
   proof -
     from A2 A3 have ∀k ∈ n. {b(k)} ∈  FinPow(X)
using bij_def inj_def apply_funtype FinPow_def
  singleton_in_finpow by auto
     then show M : n → FinPow(X) using ZF_fun_from_total
by simp
     from A3 show M {is partition} using bij_def inj_partition
by auto
   qed
   ultimately show
     (∏ {⟨i,∏(M(i),a)⟩. i ∈ n}) = (∏(⋃i ∈ n. M(i),a))
     by (rule prod_list_of_sets)
```

```
  qed
  also from A3 have (∏(⋃i ∈ n. M(i),a)) = ∏(A,a)
    using bij_def inj_partition surj_singleton_image
    by auto
  finally show thesis by simp
qed
```

Another way of expressing the fact that the product dos not depend on the order.

```
corollary (in semigr1) prod_bij_same:
  assumes f {is commutative on} G and
  A ∈ FinPow(X) A ≠ 0 and
  b ∈ bij(|A|,A) c ∈ bij(|A|,A)
  shows (∏ (a O b)) = (∏ (a O c))
  using assms prod_order_irr by simp

end
```

30 Commutative Semigroups

```
theory CommutativeSemigroup_ZF imports Semigroup_ZF

begin
```

In the `Semigroup` theory we introduced a notion of `SetFold(f,a,Λ,r)` that represents the sum of values of some function a valued in a semigroup where the arguments of that function vary over some set Λ. Using the additive notation something like this would be expressed as $\sum_{x\in\Lambda} f(x)$ in informal mathematics. This theory considers an alternative to that notion that is more specific to commutative semigroups.

30.1 Sum of a function over a set

The r parameter in the definition of `SetFold(f,a,Λ,r)` (from `Semigroup_ZF`) represents a linear order relation on Λ that is needed to indicate in what order we are summing the values $f(x)$. If the semigroup operation is commutative the order does not matter and the relation r is not needed. In this section we define a notion of summing up values of some function $a : X \to G$ over a finite set of indices $\Gamma \subseteq X$, without using any order relation on X.

We define the sum of values of a function $a : X \to G$ over a set Λ as the only element of the set of sums of lists that are bijections between the number of values in Λ (which is a natural number $n = \{0, 1, .., n-1\}$ if Λ is finite) and Λ. The notion of `Fold1(f,c)` is defined in `Semigroup_ZF` as the fold (sum) of the list c starting from the first element of that list. The intention is to use the fact that since the result of summing up a list does not depend on the order, the set {Fold1(f,a O b). b ∈ bij(|Λ|, Λ)} is a singleton and we can extract its only value by taking its union.

definition
```
  CommSetFold(f,a,Λ) = ⋃{Fold1(f,a O b). b ∈ bij(|Λ|, Λ)}
```

the next locale sets up notation for writing about summation in commutative semigroups. We define two kinds of sums. One is the sum of elements of a list (which are just functions defined on a natural number) and the second one represents a more general notion the sum of values of a semigroup valued function over some set of arguments. Since those two types of sums are different notions they are represented by different symbols. However in the presentations they are both intended to be printed as $\sum$.

locale commsemigr =

```
  fixes G f

  assumes csgassoc: f {is associative on} G

  assumes csgcomm: f {is commutative on} G

  fixes csgsum (infixl + 69)
  defines csgsum_def[simp]: x + y ≡ f⟨x,y⟩

  fixes X a
  assumes csgaisfun: a : X → G

  fixes csglistsum (∑ _ 70)
  defines csglistsum_def[simp]: ∑k ≡ Fold1(f,k)

  fixes csgsetsum (∑)
  defines csgsetsum_def[simp]: ∑(A,h) ≡ CommSetFold(f,h,A)
```

Definition of a sum of function over a set in notation defined in the commsemigr locale.

```
lemma (in commsemigr) CommSetFolddef:
  shows (∑(A,a)) = (⋃{∑ (a O b). b ∈ bij(|A|, A)})
  using CommSetFold_def by simp
```

The next lemma states that the result of a sum does not depend on the order we calculate it. This is similar to lemma prod_order_irr in the Semigroup theory, except that the semigr1 locale assumes that the domain of the function we sum up is linearly ordered, while in commsemigr we don't have this assumption.

```
lemma (in commsemigr) sum_over_set_bij:
  assumes A1: A ∈ FinPow(X) A ≠ 0 and A2: b ∈ bij(|A|,A)
  shows (∑(A,a)) = (∑ (a O b))
proof -
```

```
  have
    ∀c ∈ bij(|A|,A). ∀ d ∈ bij(|A|,A). (∑(a O c)) = (∑(a O d))
  proof -
    { fix c assume c ∈ bij(|A|,A)
      fix d assume d ∈ bij(|A|,A)
      let r = InducedRelation(converse(c), Le)
      have semigr1(G,f,A,r,restrict(a, A))
      proof -
 have semigr0(G,f) using csgassoc semigr0_def by simp
 moreover from A1 ‹c ∈ bij(|A|,A)› have IsLinOrder(A,r)
   using bij_converse_bij card_fin_is_nat
          natord_lin_on_each_nat ind_rel_pres_lin by simp
 moreover from A1 have restrict(a, A) : A → G
   using FinPow_def csgaisfun restrict_fun by simp
 ultimately show thesis using semigr1_axioms.intro semigr1_def
   by simp
      qed
      moreover have f {is commutative on} G using csgcomm
 by simp
      moreover from A1 have A ∈  FinPow(A) A ≠ 0
 using FinPow_def by auto
      moreover note ‹c ∈ bij(|A|,A)› ‹d ∈ bij(|A|,A)›
      ultimately have
 Fold1(f,restrict(a,A) O c) = Fold1(f,restrict(a,A) O d)
 by (rule semigr1.prod_bij_same)
      hence (∑ (restrict(a,A) O c)) = (∑ (restrict(a,A) O d))
 by simp
      moreover from A1 ‹c ∈ bij(|A|,A)› ‹d ∈ bij(|A|,A)›
      have
 restrict(a,A) O c = a O c and restrict(a,A) O d = a O d
 using bij_def surj_def csgaisfun FinPow_def comp_restrict
 by auto
      ultimately have (∑(a O c)) = (∑(a O d)) by simp
      } thus thesis by blast
    qed
  with A2 have (⋃{∑(a O b). b ∈ bij(|A|, A)}) = (∑ (a O b))
    by (rule singleton_comprehension)
  then show thesis using CommSetFolddef by simp
qed
```

The result of a sum is in the semigroup. Also, as the second assertion we show that every semigroup valued function generates a homomorphism between the finite subsets of a semigroup and the semigroup. Adding an element to a set coresponds to adding a value.

```
lemma (in commsemigr) sum_over_set_add_point:
  assumes  A1: A ∈ FinPow(X)  A ≠ 0
  shows ∑(A,a) ∈ G and
  ∀x ∈ X-A. ∑(A ∪ {x},a) = (∑(A,a)) + a(x)
proof -
```

```
 from A1 obtain b where b ∈ bij(|A|,A)
   using fin_bij_card by auto
 with A1 have ∑(A,a) = (∑  (a O b))
   using sum_over_set_bij by simp
 from A1 have |A| ∈ nat using card_fin_is_nat by simp
 have semigr0(G,f) using csgassoc semigr0_def by simp
 moreover
 from A1 obtain n where n ∈ nat and |A| = succ(n)
   using card_non_empty_succ by auto
 with A1  ‹b ∈ bij(|A|,A)› have
   n ∈ nat and a O b : succ(n) → G
   using bij_def inj_def FinPow_def comp_fun_subset csgaisfun
   by auto
 ultimately have Fold1(f,a O b) ∈ G by (rule semigr0.prod_type)
 with ‹∑(A,a) = (∑  (a O b))› show ∑(A,a) ∈ G
   by simp
 { fix x assume x ∈ X-A
   with A1 have (A ∪ {x}) ∈ FinPow(X) and A ∪ {x} ≠ 0
     using singleton_in_finpow union_finpow by auto
   moreover have Append(b,x) ∈ bij(|A ∪ {x}|, A ∪ {x})
   proof -
     note ‹|A| ∈ nat› ‹b ∈ bij(|A|,A)›
     moreover from ‹x ∈ X-A› have x ∉ A by simp
     ultimately have Append(b,x) ∈ bij(succ(|A|), A ∪ {x})
by (rule bij_append_point)
     with A1 ‹x ∈ X-A› show thesis
using card_fin_add_one by auto
   qed
   ultimately have (∑(A ∪ {x},a)) =  (∑  (a O Append(b,x)))
     using sum_over_set_bij by simp
   also have ... = (∑  Append(a O b, a(x)))
   proof -
     note ‹|A| ∈ nat›
     moreover
     from A1 ‹b ∈ bij(|A|, A)› have
b : |A| → A and A ⊆ X
using bij_def inj_def using FinPow_def by auto
     then have b : |A| → X by (rule func1_1_L1B)
     moreover from ‹x ∈ X-A› have x ∈ X and a : X → G
using csgaisfun by auto
     ultimately show thesis using list_compose_append
by simp
   qed
   also have ... =  (∑(A,a)) + a(x)
   proof -
     note ‹semigr0(G,f)›  ‹n ∈ nat›  ‹a O b : succ(n) → G›
     moreover from ‹x ∈ X-A› have a(x) ∈ G
using csgaisfun apply_funtype by simp
     ultimately have
```

```
 Fold1(f,Append(a O b, a(x))) = f⟨Fold1(f,a O b),a(x)⟩
 by (rule semigr0.prod_append)
      with A1 ‹b ∈ bij(|A|,A)› show thesis
 using sum_over_set_bij by simp
    qed
    finally have (∑(A ∪ {x},a)) = (∑(A,a)) + a(x)
      by simp
  } thus ∀x ∈ X-A. ∑(A ∪ {x},a) = (∑(A,a)) + a(x)
    by simp
qed

end
```

31 Monoids

theory Monoid_ZF **imports** func_ZF Loop_ZF Semigroup_ZF

begin

This theory provides basic facts about monoids.

31.1 Definition and basic properties

In this section we talk about monoids. The notion of a monoid is similar to the notion of a semigroup except that we require the existence of a neutral element. It is also similar to the notion of group except that we don't require existence of the inverse.

Monoid is a set G with an associative operation and a neutral element. The operation is a function on $G \times G$ with values in G. In the context of ZF set theory this means that it is a set of pairs $\langle x, y\rangle$, where $x \in G \times G$ and $y \in G$. In other words the operation is a certain subset of $(G \times G) \times G$. We express all this by defing a predicate `IsAmonoid(G,f)`. Here G is the "carrier" of the monoid and f is the binary operation on it.

```
definition
  IsAmonoid(G,f) ≡
  f {is associative on} G ∧
  (∃e∈G. (∀g∈G. ( (f(⟨e,g⟩) = g) ∧ (f(⟨g,e⟩) = g))))
```

The next locale called "monoid0" defines a context for theorems that concern monoids. In this contex we assume that the pair (G, f) is a monoid. We will use the $\oplus$ symbol to denote the monoid operation (for no particular reason).

```
locale monoid0 =
  fixes G f
  assumes monoidAssum: IsAmonoid(G,f)

  fixes monoper (infixl ⊕ 70)
```

```
  defines monoper_def [simp]: a ⊕ b ≡ f⟨a,b⟩
```

Propositions proven in the semigr0 locale are valid in the monoid0 locale.

```
lemma (in monoid0) semigr0_valid_in_monoid0: shows semigr0(G,f)
  using monoidAssum IsAmonoid_def semigr0_def by simp
```

The result of the monoid operation is in the monoid (carrier).

```
lemma (in monoid0) group0_1_L1:
  assumes a∈G  b∈G shows a⊕b ∈ G
  using assms monoidAssum IsAmonoid_def IsAssociative_def apply_funtype
  by auto
```

There is only one neutral element in a monoid.

```
lemma (in monoid0) group0_1_L2: shows
  ∃!e. e∈G ∧ (∀ g∈G. ( (e⊕g = g) ∧ g⊕e = g))
proof
  fix e y
  assume e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
    and y ∈ G ∧ (∀g∈G. y ⊕ g = g ∧ g ⊕ y = g)
  then have y⊕e = y y⊕e = e by auto
  thus e = y by simp
next from monoidAssum show
    ∃e. e∈ G ∧ (∀ g∈G. e⊕g = g ∧ g⊕e = g)
    using IsAmonoid_def by auto
qed
```

The neutral element is neutral.

```
lemma (in monoid0) unit_is_neutral:
  assumes A1: e = TheNeutralElement(G,f)
  shows e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
proof -
  let n = THE b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
  have ∃!b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
    using group0_1_L2 by simp
  then have n∈ G ∧ (∀ g∈G. n⊕g = g ∧ g⊕n = g)
    by (rule theI)
  with A1 show thesis
    using TheNeutralElement_def by simp
qed
```

The monoid carrier is not empty.

```
lemma (in monoid0) group0_1_L3A: shows G≠0
proof -
  have TheNeutralElement(G,f) ∈ G using unit_is_neutral
    by simp
  thus thesis by auto
qed
```

The monoid operation is a binary function on the carrier with values in the carrier.

```
lemma (in monoid0) monoid_oper_fun: shows f:G×G→G
  using monoidAssum unfolding IsAmonoid_def IsAssociative_def
  by simp
```

The range of the monoid operation is the whole monoid carrier.

```
lemma (in monoid0) group0_1_L3B: shows range(f) = G
proof
  from monoidAssum have f : G×G→G
     using IsAmonoid_def IsAssociative_def by simp
  then show range(f) ⊆ G
    using func1_1_L5B by simp
  show G ⊆ range(f)
  proof
    fix g assume A1: g∈G
    let e = TheNeutralElement(G,f)
    from A1 have ⟨e,g⟩ ∈ G×G g = f⟨e,g⟩
      using unit_is_neutral by auto
    with ‹f : G×G→G› show g ∈ range(f)
      using func1_1_L5A by blast
  qed
qed
```

Another way to state that the range of the monoid operation is the whole monoid carrier.

```
lemma (in monoid0) range_carr: shows f(G×G) = G
  using monoidAssum IsAmonoid_def IsAssociative_def
    group0_1_L3B range_image_domain by auto
```

In a monoid any neutral element is the neutral element.

```
lemma (in monoid0) group0_1_L4:
  assumes A1: e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
  shows e = TheNeutralElement(G,f)
proof -
  let n = THE b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
  have ∃!b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
    using group0_1_L2 by simp
  moreover note A1
  ultimately have n = e by (rule the_equality2)
  then show thesis using TheNeutralElement_def by simp
qed
```

The next lemma shows that if the if we restrict the monoid operation to a subset of G that contains the neutral element, then the neutral element of the monoid operation is also neutral with the restricted operation.

```
lemma (in monoid0) group0_1_L5:
```

```
  assumes A1: ∀x∈H.∀y∈H. x⊕y ∈ H
  and A2: H⊆G
  and A3: e = TheNeutralElement(G,f)
  and A4: g = restrict(f,H×H)
  and A5: e∈H
  and A6: h∈H
  shows g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
proof -
  from A4 A6 A5 have
    g⟨e,h⟩ = e⊕h ∧ g⟨h,e⟩ = h⊕e
    using restrict_if by simp
  with A3 A4 A6 A2 show
    g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
    using  unit_is_neutral by auto
qed
```

The next theorem shows that if the monoid operation is closed on a subset of G then this set is a (sub)monoid (although we do not define this notion). This fact will be useful when we study subgroups.

```
theorem (in monoid0) group0_1_T1:
  assumes A1: H {is closed under} f
  and A2: H⊆G
  and A3: TheNeutralElement(G,f) ∈ H
  shows  IsAmonoid(H,restrict(f,H×H))
proof -
  let g = restrict(f,H×H)
  let e = TheNeutralElement(G,f)
  from monoidAssum have f ∈ G×G→G
    using IsAmonoid_def IsAssociative_def by simp
  moreover from A2 have H×H ⊆ G×G by auto
  moreover from A1 have ∀p ∈ H×H. f(p) ∈ H
    using IsOpClosed_def by auto
  ultimately have g ∈ H×H→H
    using func1_2_L4 by simp
  moreover have ∀x∈H.∀y∈H.∀z∈H.
    g⟨g⟨x,y⟩ ,z⟩ = g⟨x,g⟨y,z⟩⟩
  proof -
    from A1 have ∀x∈H.∀y∈H.∀z∈H.
      g⟨g⟨x,y⟩,z⟩ = x⊕y⊕z
      using IsOpClosed_def restrict_if by simp
    moreover have ∀x∈H.∀y∈H.∀z∈H. x⊕y⊕z = x⊕(y⊕z)
    proof -
      from monoidAssum have
 ∀x∈G.∀y∈G.∀z∈G. x⊕y⊕z = x⊕(y⊕z)
 using IsAmonoid_def IsAssociative_def
 by simp
      with A2 show thesis by auto
    qed
    moreover from A1 have
```

```
      ∀x∈H.∀y∈H.∀z∈H. x⊕(y⊕z) = g⟨ x,g⟨y,z⟩ ⟩
      using IsOpClosed_def restrict_if by simp
    ultimately show thesis by simp
  qed
  moreover have
    ∃n∈H. (∀h∈H. g⟨n,h⟩ = h ∧ g⟨h,n⟩ = h)
  proof -
    from A1 have ∀x∈H.∀y∈H. x⊕y ∈ H
      using IsOpClosed_def by simp
    with A2 A3 have
      ∀ h∈H. g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
      using group0_1_L5 by blast
    with A3 show thesis by auto
  qed
  ultimately show thesis using IsAmonoid_def IsAssociative_def
    by simp
qed
```

Under the assumptions of group0_1_T1 the neutral element of a submonoid is the same as that of the monoid.

```
lemma group0_1_L6:
  assumes A1: IsAmonoid(G,f)
  and A2: H {is closed under} f
  and A3: H⊆G
  and A4: TheNeutralElement(G,f) ∈ H
  shows TheNeutralElement(H,restrict(f,H×H)) = TheNeutralElement(G,f)
proof -
  let e = TheNeutralElement(G,f)
  let g = restrict(f,H×H)
  from assms have monoid0(H,g)
    using monoid0_def monoid0.group0_1_T1
    by simp
  moreover have
    e ∈ H ∧ (∀h∈H. g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h)
  proof -
    { fix h assume h ∈ H
      with assms have
 monoid0(G,f)  ∀x∈H.∀y∈H. f⟨x,y⟩ ∈ H
 H⊆G  e = TheNeutralElement(G,f)  g = restrict(f,H×H)
 e ∈ H  h ∈ H
 using monoid0_def IsOpClosed_def by auto
      then have g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h
 by (rule monoid0.group0_1_L5)
    } hence ∀h∈H. g⟨e,h⟩ = h ∧ g⟨h,e⟩ = h by simp
    with A4 show thesis by simp
  qed
  ultimately have e =  TheNeutralElement(H,g)
    by (rule monoid0.group0_1_L4)
  thus thesis by simp
```

qed

If a sum of two elements is not zero, then at least one has to be nonzero.

```
lemma (in monoid0) sum_nonzero_elmnt_nonzero:
  assumes a ⊕ b ≠ TheNeutralElement(G,f)
  shows a ≠ TheNeutralElement(G,f) ∨ b ≠ TheNeutralElement(G,f)
  using assms unit_is_neutral by auto
```

The monoid operation is associative.

```
lemma (in monoid0) sum_associative:
  assumes a∈G b∈G c∈G
  shows (a⊕b)⊕c = a⊕(b⊕c)
  using assms monoidAssum unfolding IsAmonoid_def IsAssociative_def
  by auto
```

A simple rearrangement of four monoid elements transferred from the `semigr0` locale:

```
lemma (in monoid0) rearr4elem_monoid:
  assumes a∈G  b∈G  c∈G  d∈G
  shows a⊕b⊕(c⊕d) = a⊕(b⊕c)⊕d
  using assms semigr0_valid_in_monoid0 semigr0.rearr4elem_assoc
  by simp
```

end

32 Summing lists in a monoid

theory `Monoid_ZF_1` **imports** `Monoid_ZF`

begin

This theory consider properties of sums of monoid elements, similar to the ones formalized in the `Semigroup_ZF` theory for sums of semigroup elements. The main difference is that since each monoid has a neutral element it makes sense to define a sum of an empty list of monoid elements. In multiplicative notation the properties considered here can be applied to natural powers of elements ($x^n, n \in \mathbb{N}$) in group or ring theory or, when written additively, to natural multiplicities $n \cdot x, n \in \mathbb{N}$).

32.1 Notation and basic properties of sums of lists of monoid elements

In this section we setup a contex (locale) with notation for sums of lists of monoid elements and prove basic properties of those sums in terms of that notation.

The locale (context) `monoid1` extends the locale `monoid1`, adding the notation for the neutral element as 0 and the sum of a list of monoid elements. It also defines a notation for natural multiple of an element of a monoid, i.e. $n \cdot x = x \oplus x \oplus ... \oplus x$ (n times).

```
locale monoid1 = monoid0 +
  fixes mzero (0)
  defines mzero_def [simp]: 0 ≡ TheNeutralElement(G,f)

  fixes listsum (∑ _ 70)
  defines listsum_def [simp]: ∑s ≡ Fold(f,0,s)

  fixes nat_mult (infix · 72)
  defines nat_mult_def [simp]: n·x ≡ ∑{⟨k,x⟩. k∈n}
```

Let's recall that the neutral element of the monoid is an element of the monoid (carrier) G and the monoid operation (f in our notation) is a function that maps $G \times G$ to G.

```
lemma (in monoid1) zero_monoid_oper: shows 0∈G and f:G×G → G
  using monoidAssum unit_is_neutral unfolding IsAmonoid_def IsAssociative_def

  by simp_all
```

The sum of a list of monoid elements is a monoid element.

```
lemma (in monoid1) sum_in_mono: assumes n∈nat ∀k∈n. q(k)∈G
  shows (∑{⟨k,q(k)⟩. k∈n}) ∈ G
proof -
  let a = {⟨k,q(k)⟩. k∈n}
  from assms have n ∈ nat f:G×G → G a:n → G 0∈G G≠0
    using zero_monoid_oper ZF_fun_from_total by auto
  then show thesis using fold_props by simp
qed
```

The reason we start from 0 in the definition of the summation sign in the `monoid1` locale is that we want to be able to sum the empty list. Such sum of the empty list is 0.

```
lemma (in monoid1) sum_empty: assumes s:0→G shows (∑s) = 0
  using assms zero_monoid_oper fold_empty group0_1_L3A by simp
```

For nonempty lists our Σ is the same as `Fold1`.

```
lemma (in monoid1) sum_nonempty:  assumes n ∈ nat s:succ(n)→G
  shows
    (∑s) = Fold(f,s(0),Tail(s))
    (∑s) = Fold1(f,s)
proof -
  from assms have s(0) ∈ G using empty_in_every_succ apply_funtype
    by simp
  with assms have (∑s) = Fold(f,0⊕s(0),Tail(s))
```

```
    using zero_monoid_oper fold_detach_first by simp
  with ‹s(0) ∈ G› show (∑s) = Fold(f,s(0),Tail(s))
    using unit_is_neutral by simp
  then show (∑s) = Fold1(f,s) unfolding Fold1_def by simp
qed
```

We can pull the first component of a sum of a nonempty list of monoid elements before the summation sign.

```
lemma (in monoid1) seq_sum_pull_first0: assumes n ∈ nat s:succ(n)→G
  shows (∑s) = s(0) ⊕ (∑Tail(s))
proof -
  from assms have s(0) ∈ G using empty_in_every_succ apply_funtype
    by simp
  { assume n=0
    with assms have Tail(s):0→G using tail_props(1) by blast
    with assms ‹s(0) ∈ G› have (∑s) = s(0) ⊕ (∑Tail(s))
      using sum_nonempty(1) sum_empty zero_monoid_oper(2) group0_1_L3A

        fold_empty unit_is_neutral by simp
  }
  moreover
  { assume n≠0
    with assms(1) obtain m where m∈nat and n = succ(m)
      using Nat_ZF_1_L3 by blast
    with assms have Tail(s):succ(m)→G using tail_props(1)
      by simp
    let a = {⟨0,s(0)⟩}
    from ‹s(0) ∈ G› have 0∈nat a:succ(0)→G
      using pair_func_singleton succ_explained by simp_all
    with ‹m∈nat› ‹Tail(s):succ(m)→G›
    have f⟨Fold1(f,a),Fold1(f,Tail(s))⟩ = Fold1(f,Concat(a,Tail(s)))
      using semigr0_valid_in_monoid0 semigr0.prod_conc_distr
      by blast
    with assms ‹a:succ(0)→G› ‹m∈nat› ‹Tail(s):succ(m)→G›
    have (∑s) = s(0) ⊕ (∑Tail(s))
      using semigr0_valid_in_monoid0 semigr0.prod_of_1elem pair_val
        sum_nonempty(2) first_concat_tail by simp
  }
  ultimately show thesis by auto
qed
```

The first assertion of the next theorem is similar in content to `seq_sum_pull_first0` formulated in terms of the expression defining the list of monoid elements. The second one shows the dual statement: the last element of a sequence can be pulled out of the sequence and put after the summation sign. So, we are showing here that $\sum_{k=0}^{n} q_k = q_0 \oplus \sum_{k=0}^{n-1} q_{k+1} = (\sum_{k=0}^{n-1} q_k) \oplus q_n$.

```
theorem (in monoid1) seq_sum_pull_one_elem:
  assumes n ∈ nat ∀k∈n #+ 1. q(k) ∈ G
```

```
  shows
    (∑{⟨k,q(k)⟩. k∈n #+ 1}) = q(0) ⊕ (∑{⟨k,q(k #+ 1)⟩. k∈n})
    (∑{⟨k,q(k)⟩. k∈n #+ 1}) = (∑{⟨k,q(k)⟩. k∈n}) ⊕ q(n)
proof -
  let s = {⟨k,q(k)⟩. k∈n #+ 1}
  from assms(1) have 0 ∈ n #+ 1 using empty_in_every_succ succ_add_one(1)
    by simp
  then have s(0) = q(0) by (rule ZF_fun_from_tot_val1)
  from assms(2) have s: n #+ 1 → G by (rule ZF_fun_from_total)
  with assms(1) ‹s(0) = q(0)› have (∑s) = q(0) ⊕ (∑Tail(s))
    using seq_sum_pull_first0 by simp
  moreover from assms have Tail(s) = {⟨k,q(k #+ 1)⟩. k ∈ n}
    using tail_formula by simp
  ultimately show (∑{⟨k,q(k)⟩. k∈n #+ 1}) =  q(0) ⊕ (∑{⟨k,q(k #+ 1)⟩.
k∈n})
    by simp
  from assms show (∑{⟨k,q(k)⟩. k∈n #+ 1}) =  (∑{⟨k,q(k)⟩. k∈n}) ⊕
q(n)
    using zero_monoid_oper fold_detach_last by simp
qed
```

The sum of a singleton list is its only element,

```
lemma (in monoid1) seq_sum_singleton: assumes q(0) ∈ G
  shows (∑{⟨k,q(k)⟩. k∈1}) = q(0)
proof -
  from assms have 0∈nat and ∀k∈0 #+ 1. q(k) ∈ G by simp_all
  then have
    (∑{⟨k,q(k)⟩. k∈0 #+ 1}) = q(0) ⊕ (∑{⟨k,q(k #+ 1)⟩. k∈0})
    by (rule seq_sum_pull_one_elem)
  with assms show thesis using sum_empty unit_is_neutral by simp
qed
```

If the monoid operation is commutative, then the sum of a nonempty sequence added to another sum of a nonempty sequence of the same length is equal to the sum of pointwise sums of the sequence elements. This is the same as the theorem `prod_comm_distrib` from the `Semigroup_ZF` theory, just written in the notation used in the `monoid1` locale.

```
lemma (in monoid1) sum_comm_distrib0:
  assumes  f {is commutative on} G n∈nat and
  a : n #+ 1 → G  b : n #+ 1 → G  c : n #+ 1 → G and
  ∀j∈n #+ 1. c(j) = a(j) ⊕ b(j)
  shows (∑ c) = (∑ a) ⊕ (∑ b)
  using assms succ_add_one(1) sum_nonempty
    semigr0_valid_in_monoid0 semigr0.prod_comm_distrib by simp
```

Another version of `sum_comm_distrib0` written in terms of the expressions defining the sequences, shows that for commutative monoids we have $\sum_{k=0}^{n-1} q(k) \oplus p(k) = (\sum_{k=0}^{n-1} p(k)) \oplus (\sum_{k=0}^{n-1} q(k))$.

```
theorem (in monoid1) sum_comm_distrib:
  assumes  f {is commutative on} G n∈nat and
  ∀k∈n. p(k) ∈ G ∀k∈n. q(k) ∈ G
  shows
    (∑{⟨k,p(k)⊕q(k)⟩. k∈n}) = (∑{⟨k,p(k)⟩. k∈n}) ⊕ (∑{⟨k,q(k)⟩. k∈n})

proof -
  let a = {⟨k,p(k)⟩. k∈n}
  let b = {⟨k,q(k)⟩. k∈n}
  let c = {⟨k,p(k)⊕q(k)⟩. k∈n}
  { assume n=0
    then have (∑c) = (∑a) ⊕ (∑b)
      using sum_empty unit_is_neutral by simp
  }
  moreover
  { assume n≠0
    with assms(2) obtain m where m∈nat and  n = m #+ 1
      using nat_not0_succ by blast
    from assms(3,4) have a:n→G b:n→G c:n→G
      using group0_1_L1 ZF_fun_from_total by simp_all
    with assms(1) ‹m∈nat› ‹n = m #+ 1› have
      f {is commutative on} G m∈nat and
      a:m #+ 1→G b:m #+ 1→G c:m #+ 1→G
      by simp_all
    moreover have ∀k∈m #+ 1. c(k) = a(k) ⊕ b(k)
    proof -
      { fix k assume k ∈ m #+ 1
        with ‹n = m #+ 1› have k∈n by simp
        then have c(k) = a(k) ⊕ b(k)
          using ZF_fun_from_tot_val1 by simp_all
      } thus thesis by simp
    qed
    ultimately have (∑c) = (∑a) ⊕ (∑b)
      using sum_comm_distrib0 by simp
  }
  ultimately show thesis by blast
qed
```

32.2 Multiplying monoid elements by natural numbers

A special case of summing (or, using more notation-neutral term `folding`) a list of monoid elements is taking a natural multiple of a single element. This can be applied to various monoids embedded in other algebraic structures. For example a ring is a monoid with addition as the operation, so the notion of natural multiple directly transfers there. Another monoid in a ring is formed by its multiplication operation. In that case the natural multiple maps into natural powers of a ring element.

Another way of looking at a multiple of a monoid element: it's a sum of the cartesian product of n and the singleton $\{x\}$. This is because the expression $\{\langle k,x\rangle : k \in n\}$ in the defintion of the notation for natural multiple i.e. a constant list of the length n is the same as the set $n \times \{x\}$.

```
lemma (in monoid1) monoid_nat_mult_def_alt: shows n·x = ∑n×{x}
  using const_fun_def_alt const_fun_def_alt1 by simp
```

The zero's multiple of a monoid element is its neutral element.

```
lemma (in monoid1) nat_mult_zero: shows 0·x = 0 using sum_empty by simp
```

Any multiple of a monoid element is a monoid element.

```
lemma (in monoid1) nat_mult_type: assumes n∈nat x∈G
  shows n·x ∈ G using assms sum_in_mono by simp
```

Taking one more multiple of x adds x.

```
lemma (in monoid1) nat_mult_add_one: assumes n∈nat x∈G
  shows (n #+ 1)·x = n·x ⊕ x and (n #+ 1)·x = x ⊕ n·x
proof -
  from assms(2) have I: ∀k∈n #+ 1. x ∈ G by simp
  with assms(1) have (∑{⟨k,x⟩. k ∈ n #+ 1}) = x ⊕ (∑{⟨k,x⟩. k∈n})
    by (rule seq_sum_pull_one_elem)
  thus (n #+ 1)·x = x ⊕ n·x by simp
  from assms(1) I have (∑{⟨k,x⟩. k∈n #+ 1}) =  (∑{⟨k,x⟩. k∈n}) ⊕ x
    by (rule seq_sum_pull_one_elem)
  thus (n #+ 1)·x = n·x ⊕ x by simp
qed
```

One element of a monoid is that element.

```
lemma (in monoid1) nat_mult_one: assumes x∈G shows 1·x = x
proof -
  from assms have (0 #+ 1)·x = 0·x ⊕ x using nat_mult_add_one(1) by blast
  with assms show thesis using nat_mult_zero unit_is_neutral by simp
qed
```

Multiplication of x by a natural number induces a homomorphism between natural numbers with addition and and the natural multiples of x.

```
lemma (in monoid1) nat_mult_add: assumes n∈nat m∈nat x∈G
  shows (n #+ m)·x = n·x ⊕ m·x
proof -
  from assms have m∈nat and (n #+ 0)·x = n·x ⊕ 0·x
    using nat_mult_type unit_is_neutral nat_mult_zero by simp_all
  moreover from assms(1,3) have
    ∀k∈nat. (n #+ k)·x = n·x ⊕ k·x ⟶ (n #+ (k #+ 1))·x = n·x ⊕ (k #+
1)·x
     using nat_mult_type nat_mult_add_one(1) sum_associative by simp
  ultimately show thesis by (rule ind_on_nat1)
qed
```

The neutral element is fixed by this operation.

```
lemma (in monoid1) nat_mult_neutral: assumes n∈nat shows n·0 = 0
proof -
  from assms have n∈nat and 0·0 = 0 using nat_mult_zero by auto
  moreover have
    ∀k∈nat. k·0 = 0 ⟶ (k #+ 1)·0 = 0
    using nat_mult_add_one nat_mult_one unit_is_neutral by auto
  ultimately show thesis by (rule ind_on_nat1)
qed
```

end

33 Groups - introduction

theory Group_ZF **imports** Monoid_ZF_1 Semigroup_ZF

begin

This theory file covers basics of group theory.

33.1 Definition and basic properties of groups

In this section we define the notion of a group and set up the notation for discussing groups. We prove some basic theorems about groups.

To define a group we take a monoid and add a requirement that the right inverse needs to exist for every element of the group.

```
definition
  IsAgroup(G,f) ≡
  (IsAmonoid(G,f) ∧ (∀g∈G. ∃b∈G. f⟨g,b⟩ = TheNeutralElement(G,f)))
```

We define the group inverse as the set $\{\langle x,y\rangle \in G\times G : x\cdot y = e\}$, where e is the neutral element of the group. This set (which can be written as $(\cdot)^{-1}\{e\}$) is a certain relation on the group (carrier). Since, as we show later, for every $x\in G$ there is exactly one $y\in G$ such that $x\cdot y = e$ this relation is in fact a function from G to G.

```
definition
  GroupInv(G,f) ≡ {⟨x,y⟩ ∈ G×G. f⟨x,y⟩ = TheNeutralElement(G,f)}
```

Next we define the context `group0` in which groups will be discussed. The common assumption for all proposition proven in the context `group0` is that fixed the pair (G,P) is a group. We will use the multiplicative notation for groups. The neutral element is denoted 1. `x`$^{-1}$ will denote the inverse of x, i.e. the value of the group inverse operation on x. We define the notation for product of a finite list of elements of G using the notion of `Fold`, defined in the `Fold_ZF` theory. Finally we define the notation `pow(n,x)` which is a product of the list of length n with value x repeated n times. Such list is a function that assigns x to every element of the set $n = \{0, 1, ..., n-1\}$ and is represented by a set of pairs $\{\langle k, x\rangle : k \in n\}$, which is the same as the set $n \times \{x\}$ (see lemma fun_def_alt1 in the func1 theory.

```
locale group0 =
  fixes G
  fixes P
  assumes groupAssum: IsAgroup(G,P)

  fixes neut (1)
  defines neut_def[simp]: 1 ≡ TheNeutralElement(G,P)

  fixes groper (infixl · 70)
  defines groper_def[simp]: a · b ≡ P⟨a,b⟩

  fixes inv (_⁻¹ [90] 91)
  defines inv_def[simp]: x⁻¹ ≡ GroupInv(G,P)(x)

  fixes listprod (∏ _ 70)
  defines listprod_def [simp]: ∏s ≡ Fold(P,1,s)

  fixes pow
  defines pow_def [simp]: pow(n,x) ≡ ∏{⟨k,x⟩. k∈n}
```

First we show a lemma that says that we can use theorems proven in the monoid0 context (locale).

```
lemma (in group0) group0_2_L1: shows monoid0(G,P)
  using groupAssum IsAgroup_def monoid0_def by simp
```

Assumptions of the monoid1 context are satisfied in the group0 context.

```
lemma (in group0) monoid1_valid_in_group: shows monoid1(G,P)
  using group0_2_L1 monoid1_def by simp
```

The theorems proven in the monoid1 context are valid in the group0 context.

```
sublocale group0 < monoid: monoid1 G P groper 1 listprod pow
  using monoid1_valid_in_group by auto
```

In some strange cases Isabelle has difficulties with applying the definition of a group. The next lemma defines a rule to be applied in such cases.

```
lemma definition_of_group: assumes IsAmonoid(G,f)
  and ∀g∈G. ∃b∈G. f⟨g,b⟩ = TheNeutralElement(G,f)
  shows IsAgroup(G,f)
  using assms IsAgroup_def by simp
```

A technical lemma that allows to use 1 as the neutral element of the group without referencing a list of lemmas and definitions.

```
lemma (in group0) group0_2_L2:
  shows 1∈G ∧ (∀g∈G.(1·g = g ∧ g·1 = g))
```

```
  using group0_2_L1 monoid.unit_is_neutral by simp
```

The group is closed under the group operation. Used all the time, useful to have handy.

```
lemma (in group0) group_op_closed: assumes a∈G  b∈G
  shows a·b ∈ G using assms monoid.group0_1_L1
  by simp
```

The group operation is associative. This is another technical lemma that allows to shorten the list of referenced lemmas in some proofs.

```
lemma (in group0) group_oper_assoc:
  assumes a∈G  b∈G  c∈G shows a·(b·c) = a·b·c
  using groupAssum assms IsAgroup_def IsAmonoid_def
    IsAssociative_def group_op_closed by simp
```

The group operation maps $G \times G$ into G. It is conveniet to have this fact easily accessible in the `group0` context.

```
lemma (in group0) group_oper_fun: shows P : G×G→G
  using groupAssum IsAgroup_def IsAmonoid_def IsAssociative_def
  by simp
```

The definition of a group requires the existence of the right inverse. We show that this is also the left inverse.

```
theorem (in group0) group0_2_T1:
  assumes A1: g∈G and A2: b∈G and A3: g·b = 1
  shows b·g = 1
proof -
  from A2 groupAssum obtain c where I: c ∈ G ∧ b·c = 1
    using IsAgroup_def by auto
  then have c∈G by simp
  have 1∈G using group0_2_L2 by simp
  with A1 A2 I have b·g =  b·(g·(b·c))
    using group_op_closed group0_2_L2 group_oper_assoc
    by simp
  also from  A1 A2 ‹c∈G› have b·(g·(b·c)) = b·(g·b·c)
    using group_oper_assoc by simp
  also from A3 A2 I have b·(g·b·c)= 1 using group0_2_L2 by simp
  finally show b·g = 1 by simp
qed
```

For every element of a group there is only one inverse.

```
lemma (in group0) group0_2_L4:
  assumes A1: x∈G shows ∃!y. y∈G ∧ x·y = 1
proof
  from A1 groupAssum show ∃y. y∈G ∧  x·y = 1
    using IsAgroup_def by auto
  fix y n
  assume A2: y∈G ∧  x·y = 1 and A3:n∈G ∧ x·n = 1 show y=n
```

```
  proof -
    from A1 A2 have T1: y·x = 1
      using group0_2_T1 by simp
    from A2 A3 have y = y·(x·n)
      using group0_2_L2 by simp
    also from A1 A2 A3 have ... = (y·x)·n
      using group_oper_assoc by blast
    also from T1 A3 have ... = n
      using group0_2_L2 by simp
    finally show y=n by simp
  qed
qed
```

The group inverse is a function that maps G into G.

```
theorem group0_2_T2:
  assumes A1: IsAgroup(G,f) shows GroupInv(G,f) : G→G
proof -
  have GroupInv(G,f) ⊆ G×G using GroupInv_def by auto
  moreover from A1 have
    ∀x∈G. ∃!y. y∈G ∧ ⟨x,y⟩ ∈ GroupInv(G,f)
    using group0_def group0.group0_2_L4 GroupInv_def by simp
  ultimately show thesis using func1_1_L11 by simp
qed
```

We can think about the group inverse (the function) as the inverse image of the neutral element. Recall that in Isabelle `f-(A)` denotes the inverse image of the set A.

```
theorem (in group0) group0_2_T3: shows P-{1} = GroupInv(G,P)
proof -
  from groupAssum have P : G×G → G
    using IsAgroup_def IsAmonoid_def IsAssociative_def
    by simp
  then show P-{1} = GroupInv(G,P)
    using func1_1_L14 GroupInv_def by auto
qed
```

The inverse is in the group.

```
lemma (in group0) inverse_in_group: assumes A1: x∈G shows x⁻¹∈G
proof -
  from groupAssum have GroupInv(G,P) : G→G using group0_2_T2 by simp
  with A1 show thesis using apply_type by simp
qed
```

The notation for the inverse means what it is supposed to mean.

```
lemma (in group0) group0_2_L6:
  assumes A1: x∈G shows x·x⁻¹ = 1 ∧ x⁻¹·x = 1
proof
  from groupAssum have GroupInv(G,P) : G→G
```

```
    using group0_2_T2 by simp
  with A1 have ⟨x,x⁻¹⟩ ∈  GroupInv(G,P)
    using apply_Pair by simp
  then show x·x⁻¹ = 1 using GroupInv_def by simp
  with A1 show x⁻¹·x = 1 using inverse_in_group group0_2_T1
    by blast
qed
```

The next two lemmas state that unless we multiply by the neutral element, the result is always different than any of the operands.

```
lemma (in group0) group0_2_L7:
  assumes A1: a∈G and A2: b∈G and A3: a·b = a
  shows b=1
proof -
  from A3 have a⁻¹ · (a·b) = a⁻¹·a by simp
  with A1 A2 show thesis using
    inverse_in_group group_oper_assoc group0_2_L6 group0_2_L2
    by simp
qed
```

See the comment to group0_2_L7.

```
lemma (in group0) group0_2_L8:
  assumes A1: a∈G and A2: b∈G and A3: a·b = b
  shows a=1
proof -
  from A3 have (a·b)·b⁻¹  = b·b⁻¹ by simp
  with A1 A2 have a·(b·b⁻¹)  = b·b⁻¹ using
    inverse_in_group group_oper_assoc by simp
  with A1 A2 show thesis
    using group0_2_L6 group0_2_L2 by simp
qed
```

The inverse of the neutral element is the neutral element.

```
lemma (in group0) group_inv_of_one: shows 1⁻¹ = 1
  using group0_2_L2 inverse_in_group group0_2_L6 group0_2_L7 by blast
```

Dividing by the neutral element does not change the dividend.

```
lemma (in group0) div_by_neutral: assumes x∈G shows x·1⁻¹ = x
  using assms group_inv_of_one group0_2_L2 by simp
```

if $a^{-1} = 1$, then $a = 1$.

```
lemma (in group0) group0_2_L8A:
  assumes A1: a∈G and A2: a⁻¹ = 1
  shows a = 1
proof -
  from A1 have a·a⁻¹ = 1 using group0_2_L6 by simp
  with A1 A2 show a = 1 using group0_2_L2 by simp
qed
```

If a is not a unit, then its inverse is not a unit either.

```
lemma (in group0) group0_2_L8B:
  assumes a∈G and a ≠ 1
  shows a⁻¹ ≠ 1 using assms group0_2_L8A by auto
```

If a^{-1} is not a unit, then a is not a unit either.

```
lemma (in group0) group0_2_L8C:
  assumes a∈G and a⁻¹ ≠ 1
  shows a≠1
  using assms group0_2_L8A group_inv_of_one by auto
```

If a product of two elements of a group is equal to the neutral element then they are inverses of each other.

```
lemma (in group0) group0_2_L9:
  assumes A1: a∈G and A2: b∈G and A3: a·b = 1
  shows a = b⁻¹ and b = a⁻¹
proof -
  from A3 have a·b·b⁻¹ = 1·b⁻¹ by simp
  with A1 A2 have a·(b·b⁻¹) = 1·b⁻¹ using
    inverse_in_group group_oper_assoc by simp
  with A1 A2 show a = b⁻¹ using
    group0_2_L6 inverse_in_group group0_2_L2 by simp
  from A3 have a⁻¹·(a·b) = a⁻¹·1 by simp
  with A1 A2 show b = a⁻¹ using
    inverse_in_group group_oper_assoc group0_2_L6 group0_2_L2
    by simp
qed
```

It happens quite often that we know what is (have a meta-function for) the right inverse in a group. The next lemma shows that the value of the group inverse (function) is equal to the right inverse (meta-function).

```
lemma (in group0) group0_2_L9A:
  assumes A1: ∀g∈G. b(g) ∈ G ∧ g·b(g) = 1
  shows ∀g∈G. b(g) = g⁻¹
proof
  fix g assume g∈G
  moreover from A1 ‹g∈G› have b(g) ∈ G by simp
  moreover from A1 ‹g∈G› have g·b(g) = 1 by simp
  ultimately show b(g) = g⁻¹ by (rule group0_2_L9)
qed
```

What is the inverse of a product?

```
lemma (in group0) group_inv_of_two:
  assumes A1: a∈G and A2: b∈G
  shows  b⁻¹·a⁻¹ = (a·b)⁻¹
proof -
  from A1 A2 have
    b⁻¹∈G  a⁻¹∈G  a·b∈G  b⁻¹·a⁻¹ ∈ G
```

```
    using inverse_in_group group_op_closed
    by auto
  from A1 A2 ‹b⁻¹·a⁻¹ ∈ G›  have a·b·(b⁻¹·a⁻¹) = a·(b·(b⁻¹·a⁻¹))
    using group_oper_assoc by simp
  moreover from A2 ‹b⁻¹∈G› ‹a⁻¹∈G› have b·(b⁻¹·a⁻¹) = b·b⁻¹·a⁻¹
    using group_oper_assoc by simp
  moreover from A2 ‹a⁻¹∈G› have b·b⁻¹·a⁻¹ = a⁻¹
     using group0_2_L6 group0_2_L2 by simp
  ultimately have a·b·(b⁻¹·a⁻¹) = a·a⁻¹
    by simp
  with A1 have a·b·(b⁻¹·a⁻¹) = 1
    using group0_2_L6 by simp
  with ‹a·b ∈ G›  ‹b⁻¹·a⁻¹ ∈ G› show b⁻¹·a⁻¹ = (a·b)⁻¹
    using group0_2_L9 by simp
qed
```

What is the inverse of a product of three elements?

```
lemma (in group0) group_inv_of_three:
  assumes A1: a∈G  b∈G  c∈G
  shows
  (a·b·c)⁻¹ = c⁻¹·(a·b)⁻¹
  (a·b·c)⁻¹ = c⁻¹·(b⁻¹·a⁻¹)
  (a·b·c)⁻¹ = c⁻¹·b⁻¹·a⁻¹
proof -
  from A1 have T:
    a·b ∈ G  a⁻¹ ∈ G  b⁻¹ ∈ G   c⁻¹ ∈ G
    using group_op_closed inverse_in_group by auto
  with A1 show
    (a·b·c)⁻¹ = c⁻¹·(a·b)⁻¹ and (a·b·c)⁻¹ = c⁻¹·(b⁻¹·a⁻¹)
     using group_inv_of_two by auto
   with T show (a·b·c)⁻¹ = c⁻¹·b⁻¹·a⁻¹ using group_oper_assoc
     by simp
qed
```

The inverse of the inverse is the element.

```
lemma (in group0) group_inv_of_inv:
  assumes a∈G shows a = (a⁻¹)⁻¹
  using assms inverse_in_group group0_2_L6 group0_2_L9
  by simp
```

Group inverse is nilpotent, therefore a bijection and involution.

```
lemma (in group0) group_inv_bij:
  shows GroupInv(G,P) O GroupInv(G,P) = id(G) and GroupInv(G,P) ∈ bij(G,G)
and
  GroupInv(G,P) = converse(GroupInv(G,P))
proof -
  have I: GroupInv(G,P): G→G using groupAssum group0_2_T2 by simp
  then have GroupInv(G,P) O GroupInv(G,P): G→G and id(G):G→G
    using comp_fun id_type by auto
```

```
  moreover
  { fix g assume g∈G
    with I have (GroupInv(G,P) O GroupInv(G,P))(g) = id(G)(g)
      using comp_fun_apply group_inv_of_inv id_conv by simp
  } hence ∀g∈G. (GroupInv(G,P) O GroupInv(G,P))(g) = id(G)(g) by simp
  ultimately show GroupInv(G,P) O GroupInv(G,P) = id(G)
    by (rule func_eq)
  with I show GroupInv(G,P) ∈ bij(G,G) using nilpotent_imp_bijective
    by simp
  with ‹GroupInv(G,P) O GroupInv(G,P) = id(G)› show
    GroupInv(G,P) = converse(GroupInv(G,P)) using comp_id_conv by simp
qed
```

A set comprehension form of the image of a set under the group inverse.

```
lemma (in group0) ginv_image: assumes V⊆G
  shows GroupInv(G,P)(V) ⊆ G and GroupInv(G,P)(V) = {g⁻¹. g ∈ V}
proof -
  from assms have I: GroupInv(G,P)(V) = {GroupInv(G,P)(g). g∈V}
    using groupAssum group0_2_T2 func_imagedef by blast
  thus GroupInv(G,P)(V) = {g⁻¹. g ∈ V}  by simp
  show GroupInv(G,P)(V) ⊆ G using groupAssum group0_2_T2 func1_1_L6(2)
by blast
qed
```

Inverse of an element that belongs to the inverse of the set belongs to the set.

```
lemma (in group0) ginv_image_el: assumes V⊆G g ∈ GroupInv(G,P)(V)
  shows g⁻¹ ∈ V
  using assms ginv_image group_inv_of_inv by auto
```

For the group inverse the image is the same as inverse image.

```
lemma (in group0) inv_image_vimage: shows GroupInv(G,P)(V) = GroupInv(G,P)-(V)
  using group_inv_bij vimage_converse by simp
```

If the unit is in a set then it is in the inverse of that set.

```
lemma (in group0) neut_inv_neut: assumes A⊆G and 1∈A
  shows 1 ∈ GroupInv(G,P)(A)
proof -
  have GroupInv(G,P):G→G using groupAssum group0_2_T2 by simp
  with assms have 1⁻¹ ∈ GroupInv(G,P)(A) using func_imagedef by auto
  then show thesis using group_inv_of_one by simp
qed
```

The group inverse is onto.

```
lemma (in group0) group_inv_surj: shows GroupInv(G,P)(G) = G
  using group_inv_bij bij_def surj_range_image_domain by auto
```

If $a^{-1} \cdot b = 1$, then $a = b$.

```
lemma (in group0) group0_2_L11:
  assumes A1: a∈G  b∈G and A2: a⁻¹·b = 1
  shows a=b
proof -
  from A1 A2 have a⁻¹ ∈ G  b∈G  a⁻¹·b = 1
    using inverse_in_group by auto
  then have b = (a⁻¹)⁻¹ by (rule group0_2_L9)
  with A1 show a=b using group_inv_of_inv by simp
qed
```

If $a \cdot b^{-1} = 1$, then $a = b$.

```
lemma (in group0) group0_2_L11A:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ = 1
  shows a=b
proof -
  from A1 A2 have a ∈ G  b⁻¹∈G  a·b⁻¹ = 1
    using inverse_in_group by auto
  then have a = (b⁻¹)⁻¹ by (rule group0_2_L9)
  with A1 show a=b using group_inv_of_inv by simp
qed
```

Inverses of different group elements are different.

```
lemma (in group0) el_neq_inv_neq: assumes a∈G  b∈G a≠b
  shows a⁻¹ ≠ b⁻¹
proof -
  { assume a⁻¹ = b⁻¹
    hence (a⁻¹)⁻¹ = (b⁻¹)⁻¹ by simp
    with assms have False using group_inv_of_inv by simp
  } thus a⁻¹ ≠ b⁻¹ by auto
qed
```

If if the inverse of b is different than a, then the inverse of a is different than b.

```
lemma (in group0) group0_2_L11B:
  assumes A1: a∈G and A2: b⁻¹ ≠ a
  shows a⁻¹ ≠ b
proof -
  { assume a⁻¹ = b
    then have (a⁻¹)⁻¹ = b⁻¹ by simp
    with A1 A2 have False using group_inv_of_inv
      by simp
  } then show a⁻¹ ≠ b by auto
qed
```

What is the inverse of ab^{-1} ?

```
lemma (in group0) group0_2_L12:
  assumes A1: a∈G  b∈G
  shows
```

```
  (a·b⁻¹)⁻¹ = b·a⁻¹
  (a⁻¹·b)⁻¹ = b⁻¹·a
proof -
  from A1 have
    (a·b⁻¹)⁻¹ = (b⁻¹)⁻¹· a⁻¹ and (a⁻¹·b)⁻¹ = b⁻¹·(a⁻¹)⁻¹
    using inverse_in_group group_inv_of_two by auto
  with A1 show  (a·b⁻¹)⁻¹ = b·a⁻¹  (a⁻¹·b)⁻¹ = b⁻¹·a
    using group_inv_of_inv by auto
qed
```

A couple useful rearrangements with three elements: we can insert a $b \cdot b^{-1}$ between two group elements (another version) and one about a product of an element and inverse of a product, and two others.

```
lemma (in group0) group0_2_L14A:
  assumes A1: a∈G  b∈G  c∈G
  shows
  a·c⁻¹= (a·b⁻¹)·(b·c⁻¹)
  a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
  a·(b·c)⁻¹ = a·c⁻¹·b⁻¹
  a·(b·c⁻¹) = a·b·c⁻¹
  (a·b⁻¹·c⁻¹)⁻¹ = c·b·a⁻¹
  a·b·c⁻¹·(c·b⁻¹) = a
  a·(b·c)·c⁻¹ = a·b
proof -
  from A1 have T:
    a⁻¹ ∈ G  b⁻¹∈G  c⁻¹∈G
    a⁻¹·b ∈ G  a·b⁻¹ ∈ G  a·b ∈ G
    c·b⁻¹ ∈ G  b·c ∈ G
    using inverse_in_group group_op_closed
    by auto
   from A1 T have
     a·c⁻¹ =  a·(b⁻¹·b)·c⁻¹
     a⁻¹·c =  a⁻¹·(b·b⁻¹)·c
    using group0_2_L2 group0_2_L6 by auto
   with A1 T show
     a·c⁻¹= (a·b⁻¹)·(b·c⁻¹)
     a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
     using group_oper_assoc by auto
  from A1 have a·(b·c)⁻¹ = a·(c⁻¹·b⁻¹)
    using group_inv_of_two by simp
  with A1 T show a·(b·c)⁻¹ =a·c⁻¹·b⁻¹
    using group_oper_assoc by simp
  from A1 T show a·(b·c⁻¹) = a·b·c⁻¹
    using group_oper_assoc by simp
  from A1 T show  (a·b⁻¹·c⁻¹)⁻¹ = c·b·a⁻¹
    using group_inv_of_three  group_inv_of_inv
    by simp
  from T have a·b·c⁻¹·(c·b⁻¹) = a·b·(c⁻¹·(c·b⁻¹))
    using group_oper_assoc by simp
```

```
  also from A1 T have ... =  a·b·b⁻¹
    using group_oper_assoc group0_2_L6 group0_2_L2
    by simp
  also from A1 T have ... = a·(b·b⁻¹)
    using group_oper_assoc by simp
  also from A1 have ... = a
    using group0_2_L6 group0_2_L2 by simp
  finally show a·b·c⁻¹·(c·b⁻¹) = a by simp
  from A1 T have a·(b·c)·c⁻¹ =  a·(b·(c·c⁻¹))
    using group_oper_assoc by simp
  also from A1 T have ... = a·b
    using  group0_2_L6 group0_2_L2 by simp
  finally show a·(b·c)·c⁻¹ = a·b
    by simp
qed
```

A simple equation to solve

```
lemma (in group0) simple_equation0:
  assumes a∈G  b∈G c∈G a·b⁻¹ = c⁻¹
  shows c = b·a⁻¹
proof -
  from assms(4) have (a·b⁻¹)⁻¹ = (c⁻¹)⁻¹ by simp
  with assms(1,2,3) show c = b·a⁻¹ using group0_2_L12(1) group_inv_of_inv
by simp
qed
```

Another simple equation

```
lemma (in group0) simple_equation1:
  assumes a∈G  b∈G c∈G a⁻¹·b = c⁻¹
  shows c = b⁻¹·a
proof -
  from assms(4) have (a⁻¹·b)⁻¹ = (c⁻¹)⁻¹ by simp
  with assms(1,2,3) show c = b⁻¹·a using group0_2_L12(2) group_inv_of_inv
by simp
qed
```

Another lemma about rearranging a product of four group elements.

```
lemma (in group0) group0_2_L15:
  assumes A1: a∈G  b∈G  c∈G  d∈G
  shows (a·b)·(c·d)⁻¹ = a·(b·d⁻¹)·a⁻¹·(a·c⁻¹)
proof -
  from A1 have T1:
    d⁻¹∈G  c⁻¹∈G a·b∈G a·(b·d⁻¹)∈G
    using inverse_in_group group_op_closed
    by auto
  with A1 have (a·b)·(c·d)⁻¹ = (a·b)·(d⁻¹·c⁻¹)
    using group_inv_of_two by simp
  also from A1 T1 have ... = a·(b·d⁻¹)·c⁻¹
    using group_oper_assoc by simp
```

```
  also from A1 T1 have ... = a·(b·d⁻¹)·a⁻¹·(a·c⁻¹)
    using group0_2_L14A by blast
  finally show thesis by simp
qed
```

We can cancel an element with its inverse that is written next to it.

```
lemma (in group0) inv_cancel_two:
  assumes A1: a∈G  b∈G
  shows
  a·b⁻¹·b = a
  a·b·b⁻¹ = a
  a⁻¹·(a·b) = b
  a·(a⁻¹·b) = b
proof -
  from A1 have
    a·b⁻¹·b = a·(b⁻¹·b)   a·b·b⁻¹ = a·(b·b⁻¹)
    a⁻¹·(a·b) = a⁻¹·a·b   a·(a⁻¹·b) = a·a⁻¹·b
    using inverse_in_group group_oper_assoc by auto
  with A1 show
    a·b⁻¹·b = a
    a·b·b⁻¹ = a
    a⁻¹·(a·b) = b
    a·(a⁻¹·b) = b
    using group0_2_L6 group0_2_L2 by auto
qed
```

Another lemma about cancelling with two group elements.

```
lemma (in group0) group0_2_L16A:
  assumes A1: a∈G  b∈G
  shows a·(b·a)⁻¹ = b⁻¹
proof -
  from A1 have (b·a)⁻¹ = a⁻¹·b⁻¹  b⁻¹ ∈ G
    using group_inv_of_two inverse_in_group by auto
  with A1 show a·(b·a)⁻¹ = b⁻¹ using inv_cancel_two
    by simp
qed
```

Some other identities with three element and cancelling.

```
lemma (in group0) cancel_middle:
  assumes a∈G  b∈G c∈G
  shows
    (a·b)⁻¹·(a·c) = b⁻¹·c
    (a·b)·(c·b)⁻¹ = a·c⁻¹
    a⁻¹·(a·b·c)·c⁻¹ = b
    a·(b·c⁻¹)·c = a·b
    a·b⁻¹·(b·c⁻¹) = a·c⁻¹
proof -
  from assms have (a·b)⁻¹·(a·c) = b⁻¹·(a⁻¹·(a·c))
```

```
    using group_inv_of_two inverse_in_group group_oper_assoc group_op_closed
by auto
  with assms(1,3) show (a·b)⁻¹·(a·c) = b⁻¹·c using inv_cancel_two(3) by
simp
  from assms have (a·b)·(c·b)⁻¹ = a·(b·(b⁻¹·c⁻¹))
    using group_inv_of_two inverse_in_group group_oper_assoc group_op_closed
by auto
  with assms show (a·b)·(c·b)⁻¹ =a·c⁻¹ using inverse_in_group inv_cancel_two(4)
by simp
  from assms have a⁻¹·(a·b·c)·c⁻¹ = (a⁻¹·a)·b·(c·c⁻¹)
    using inverse_in_group group_oper_assoc group_op_closed by auto
  with assms show a⁻¹·(a·b·c)·c⁻¹ = b using group0_2_L6 group0_2_L2 by
simp
  from assms have a·(b·c⁻¹)·c = a·b·(c⁻¹·c) using inverse_in_group group_oper_assoc
group_op_closed
    by simp
  with assms show a·(b·c⁻¹)·c = a·b using group_op_closed group0_2_L6 group0_2_L2
by simp
  from assms have a·b⁻¹·(b·c⁻¹) = a·(b⁻¹·b)·c⁻¹ using inverse_in_group group_oper_assoc
group_op_closed
    by simp
  with assms  show a·b⁻¹·(b·c⁻¹) = a·c⁻¹ using group0_2_L6 group0_2_L2
by simp
qed
```

Adding a neutral element to a set that is closed under the group operation results in a set that is closed under the group operation.

```
lemma (in group0) group0_2_L17:
  assumes H⊆G
  and H {is closed under} P
  shows (H ∪ {1}) {is closed under} P
  using assms IsOpClosed_def group0_2_L2 by auto
```

We can put an element on the other side of an equation.

```
lemma (in group0) group0_2_L18:
  assumes A1: a∈G  b∈G
  and A2: c = a·b
  shows c·b⁻¹ = a  a⁻¹·c = b
proof-
  from A2 A1 have c·b⁻¹ =  a·(b·b⁻¹)  a⁻¹·c = (a⁻¹·a)·b
    using inverse_in_group group_oper_assoc by auto
  moreover from A1 have a·(b·b⁻¹) = a  (a⁻¹·a)·b = b
    using group0_2_L6 group0_2_L2 by auto
  ultimately show c·b⁻¹ = a  a⁻¹·c = b
    by auto
qed
```

We can cancel an element on the right from both sides of an equation.

```
lemma (in group0) cancel_right: assumes a∈G  b∈G  c∈G a·b = c·b
```

```
  shows a = c
proof -
  from assms(4) have a·b·b⁻¹ = c·b·b⁻¹ by simp
  with assms(1,2,3) show thesis using inv_cancel_two(2) by simp
qed
```

We can cancel an element on the left from both sides of an equation.

```
lemma (in group0) cancel_left: assumes a∈G  b∈G  c∈G a·b = a·c
  shows b=c
proof -
  from assms(4) have a⁻¹·(a·b) = a⁻¹·(a·c) by simp
  with assms(1,2,3) show thesis using inv_cancel_two(3) by simp
qed
```

Multiplying different group elements by the same factor results in different group elements.

```
lemma (in group0) group0_2_L19:
  assumes A1: a∈G  b∈G  c∈G and A2: a≠b
  shows a·c ≠ b·c and c·a ≠ c·b
proof -
  { assume a·c = b·c ∨ c·a =c·b
    then have a·c·c⁻¹ = b·c·c⁻¹ ∨ c⁻¹·(c·a) = c⁻¹·(c·b)
      by auto
    with A1 A2 have False using inv_cancel_two by simp
  } then show a·c ≠ b·c and c·a ≠ c·b by auto
qed
```

33.2 Subgroups

There are two common ways to define subgroups. One requires that the group operation is closed in the subgroup. The second one defines subgroup as a subset of a group which is itself a group under the group operations. We use the second approach because it results in shorter definition.

The rest of this section is devoted to proving the equivalence of these two definitions of the notion of a subgroup.

A pair (H, P) is a subgroup if H forms a group with the operation P restricted to $H \times H$. It may be surprising that we don't require H to be a subset of G. This however can be inferred from the definition if the pair (G, P) is a group, see lemma group0_3_L2.

```
definition
  IsAsubgroup(H,P) ≡ IsAgroup(H, restrict(P,H×H))
```

The group is its own subgroup.

```
lemma (in group0) group_self_subgroup: shows IsAsubgroup(G,P)
  using groupAssum group_oper_fun restrict_domain
  unfolding IsAsubgroup_def by simp
```

Formally the group operation in a subgroup is different than in the group as they have different domains. Of course we want to use the original operation with the associated notation in the subgroup. The next couple of lemmas will allow for that.

The next lemma states that the neutral element of a subgroup is in the subgroup and it is both right and left neutral there. The notation is very ugly because we don't want to introduce a separate notation for the subgroup operation.

```
lemma group0_3_L1:
  assumes A1: IsAsubgroup(H,f)
  and A2: n = TheNeutralElement(H,restrict(f,H×H))
  shows n ∈ H
  ∀h∈H. restrict(f,H×H)⟨n,h ⟩ = h
  ∀h∈H. restrict(f,H×H)⟨h,n⟩ = h
proof -
  let b = restrict(f,H×H)
  let e = TheNeutralElement(H,restrict(f,H×H))
  from A1 have group0(H,b)
    using IsAsubgroup_def group0_def by simp
  then have I:
    e ∈ H ∧ (∀h∈H. (b⟨e,h ⟩ = h ∧ b⟨h,e⟩ = h))
    by (rule group0.group0_2_L2)
  with A2 show n ∈ H by simp
  from A2 I show ∀h∈H. b⟨n,h⟩ = h and ∀h∈H. b⟨h,n⟩ = h
    by auto
qed
```

A subgroup is contained in the group.

```
lemma (in group0) group0_3_L2:
  assumes A1: IsAsubgroup(H,P)
  shows H ⊆ G
proof
  fix h assume h∈H
  let b = restrict(P,H×H)
  let n = TheNeutralElement(H,restrict(P,H×H))
   from A1 have b ∈ H×H→H
    using IsAsubgroup_def IsAgroup_def
      IsAmonoid_def IsAssociative_def by simp
  moreover from A1 ‹h∈H› have ⟨ n,h⟩ ∈ H×H
    using group0_3_L1 by simp
  moreover from A1 ‹h∈H› have h = b⟨n,h ⟩
    using group0_3_L1 by simp
  ultimately have ⟨⟨n,h⟩,h⟩ ∈ b
    using func1_1_L5A by blast
  then have ⟨⟨n,h⟩,h⟩ ∈ P using restrict_subset by auto
  moreover from groupAssum have P:G×G→G
    using IsAgroup_def IsAmonoid_def IsAssociative_def
    by simp
```

```
  ultimately show h∈G using func1_1_L5
    by blast
qed
```

The group's neutral element (denoted 1 in the group0 context) is a neutral element for the subgroup with respect to the group action.

```
lemma (in group0) group0_3_L3:
  assumes IsAsubgroup(H,P)
  shows ∀h∈H. 1·h = h ∧ h·1 = h
  using assms groupAssum group0_3_L2 group0_2_L2
  by auto
```

The neutral element of a subgroup is the same as that of the group.

```
lemma (in group0) group0_3_L4: assumes A1: IsAsubgroup(H,P)
  shows TheNeutralElement(H,restrict(P,H×H)) = 1
proof -
  let n = TheNeutralElement(H,restrict(P,H×H))
  from A1 have n ∈ H using group0_3_L1 by simp
  with groupAssum A1 have n∈G using  group0_3_L2 by auto
  with A1 ‹n ∈ H› show thesis using
     group0_3_L1 restrict_if group0_2_L7 by simp
qed
```

The neutral element of the group (denoted 1 in the group0 context) belongs to every subgroup.

```
lemma (in group0) group0_3_L5: assumes A1: IsAsubgroup(H,P)
  shows 1 ∈ H
proof -
  from A1 show 1∈H using group0_3_L1 group0_3_L4
    by fast
qed
```

Subgroups are closed with respect to the group operation.

```
lemma (in group0) group0_3_L6: assumes A1: IsAsubgroup(H,P)
  and A2: a∈H b∈H
  shows a·b ∈ H
proof -
  let f = restrict(P,H×H)
  from A1 have monoid0(H,f) using
    IsAsubgroup_def IsAgroup_def monoid0_def by simp
  with A2 have f (⟨a,b⟩) ∈ H using monoid0.group0_1_L1
    by blast
 with A2 show a·b ∈ H using restrict_if by simp
qed
```

A preliminary lemma that we need to show that taking the inverse in the subgroup is the same as taking the inverse in the group.

```
lemma group0_3_L7A:
```

```
  assumes A1: IsAgroup(G,f)
  and A2: IsAsubgroup(H,f) and A3: g = restrict(f,H×H)
  shows GroupInv(G,f) ∩ H×H = GroupInv(H,g)
proof -
  let e = TheNeutralElement(G,f)
  let e1 = TheNeutralElement(H,g)
  from A1 have group0(G,f) using group0_def by simp
  from A2 A3 have group0(H,g)
    using IsAsubgroup_def group0_def by simp
  from ‹group0(G,f)› A2 A3  have GroupInv(G,f) = f-{e1}
    using group0.group0_3_L4 group0.group0_2_T3
    by simp
  moreover have g-{e1} = f-{e1} ∩ H×H
  proof -
    from A1 have f ∈ G×G→G
      using IsAgroup_def IsAmonoid_def IsAssociative_def
      by simp
    moreover from A2 ‹group0(G,f)› have H×H ⊆ G×G
      using group0.group0_3_L2 by auto
    ultimately show g-{e1} = f-{e1} ∩ H×H
      using A3 func1_2_L1 by simp
  qed
  moreover from A3 ‹group0(H,g)› have GroupInv(H,g) = g-{e1}
    using group0.group0_2_T3 by simp
  ultimately show thesis by simp
qed
```

Using the lemma above we can show the actual statement: taking the inverse in the subgroup is the same as taking the inverse in the group.

```
theorem (in group0) group0_3_T1:
  assumes A1: IsAsubgroup(H,P)
  and A2: g = restrict(P,H×H)
  shows GroupInv(H,g) = restrict(GroupInv(G,P),H)
proof -
  from groupAssum have GroupInv(G,P) : G→G
    using group0_2_T2 by simp
  moreover from A1 A2 have GroupInv(H,g) : H→H
    using IsAsubgroup_def group0_2_T2 by simp
  moreover from A1 have H ⊆ G
    using group0_3_L2 by simp
  moreover from groupAssum A1 A2 have
    GroupInv(G,P) ∩ H×H = GroupInv(H,g)
    using group0_3_L7A by simp
  ultimately show thesis
    using func1_2_L3 by simp
qed
```

A sligtly weaker, but more convenient in applications, reformulation of the above theorem.

```
theorem (in group0) group0_3_T2:
  assumes IsAsubgroup(H,P)
  and g = restrict(P,H×H)
  shows ∀h∈H. GroupInv(H,g)(h) = h⁻¹
  using assms group0_3_T1 restrict_if by simp
```

Subgroups are closed with respect to taking the group inverse.

```
theorem (in group0) group0_3_T3A:
  assumes A1: IsAsubgroup(H,P) and A2: h∈H
  shows h⁻¹∈ H
proof -
  let g = restrict(P,H×H)
  from A1 have  GroupInv(H,g) ∈ H→H
    using IsAsubgroup_def group0_2_T2 by simp
  with A2 have GroupInv(H,g)(h) ∈ H
    using apply_type by simp
  with A1 A2 show h⁻¹∈ H using group0_3_T2 by simp
qed
```

The next theorem states that a nonempty subset of a group G that is closed under the group operation and taking the inverse is a subgroup of the group.

```
theorem (in group0) group0_3_T3:
  assumes A1: H≠0
  and A2: H⊆G
  and A3: H {is closed under} P
  and A4: ∀x∈H. x⁻¹ ∈ H
  shows IsAsubgroup(H,P)
proof -
  let g = restrict(P,H×H)
  let n = TheNeutralElement(H,g)
  from A3 have I: ∀x∈H.∀y∈H. x·y ∈ H
    using IsOpClosed_def by simp
  from A1 obtain x where x∈H by auto
  with A4 I A2 have 1∈H
    using group0_2_L6 by blast
  with A3 A2 have T2: IsAmonoid(H,g)
    using monoid.group0_1_T1
    by simp
  moreover have ∀h∈H.∃b∈H. g⟨h,b⟩ = n
  proof
    fix h assume h∈H
    with A4 A2 have h·h⁻¹ = 1
      using group0_2_L6 by auto
    moreover from groupAssum A2 A3 ‹1∈H› have 1 = n
      using IsAgroup_def group0_1_L6 by auto
    moreover from A4 ‹h∈H› have g⟨h,h⁻¹⟩ = h·h⁻¹
      using restrict_if by simp
    ultimately have g⟨h,h⁻¹⟩ = n by simp
    with A4 ‹h∈H› show ∃b∈H. g⟨h,b⟩ = n by auto
```

```
  qed
  ultimately show IsAsubgroup(H,P) using
    IsAsubgroup_def IsAgroup_def by simp
qed
```

The singleton with the neutral element is a subgroup.

```
corollary (in group0) unit_singl_subgr:
  shows IsAsubgroup({1},P)
  using group0_2_L2 group_inv_of_one group0_3_T3
  unfolding IsOpClosed_def
  by auto
```

Intersection of subgroups is a subgroup. This lemma is obsolete and should be replaced by subgroup_inter.

```
lemma group0_3_L7:
  assumes A1: IsAgroup(G,f)
  and A2: IsAsubgroup(H₁,f)
  and A3: IsAsubgroup(H₂,f)
  shows IsAsubgroup(H₁∩H₂,restrict(f,H₁×H₁))
proof -
  let e = TheNeutralElement(G,f)
  let g = restrict(f,H₁×H₁)
  from A1 have I: group0(G,f)
    using group0_def by simp
  from A2 have group0(H₁,g)
    using IsAsubgroup_def group0_def by simp
  moreover have H₁∩H₂ ≠ 0
  proof -
    from A1 A2 A3 have e ∈ H₁∩H₂
      using group0_def group0.group0_3_L5 by simp
    thus thesis by auto
  qed
  moreover have H₁∩H₂ ⊆ H₁ by auto
  moreover from A2 A3 I ‹H₁∩H₂ ⊆ H₁› have
    H₁∩H₂ {is closed under} g
    using group0.group0_3_L6 IsOpClosed_def
      func_ZF_4_L7 func_ZF_4_L5 by simp
  moreover from A2 A3 I have
    ∀x ∈ H₁∩H₂. GroupInv(H₁,g)(x) ∈ H₁∩H₂
    using group0.group0_3_T2 group0.group0_3_T3A
    by simp
  ultimately show thesis
    using group0.group0_3_T3 by simp
qed
```

Intersection of subgroups is a subgroup.

```
lemma (in group0) subgroup_inter: assumes ℋ≠0
  and ∀H∈ℋ. IsAsubgroup(H,P)
  shows IsAsubgroup(⋂ℋ,P)
```

```
proof -
  {
    fix H assume H:ℋ
    with assms(2) have 1:H using group0_3_L5 by auto
  }
  then have ⋂ℋ ≠ 0 using assms(1) by auto moreover
  {
    fix t assume t:⋂ℋ
    then have ∀H∈ℋ. t:H by auto
    with assms have t:G using group0_3_L2 by blast
  }
  then have ⋂ℋ ⊆ G by auto moreover
  {
    fix x y assume xy:x:⋂ℋ y:⋂ℋ
    {
      fix J assume J:J:ℋ
      with xy have x:J y:J by auto
      with J have P⟨x,y⟩:J using assms(2) group0_3_L6 by auto
    }
    then have P⟨x,y⟩:⋂ℋ using assms(1) by auto
  }
  then have ⋂ℋ {is closed under} P unfolding IsOpClosed_def by simp
  moreover
  {
    fix x assume x:x:⋂ℋ
    {
      fix J assume J:J:ℋ
      with x have x:J by auto
      with J assms(2) have x⁻¹ ∈ J using group0_3_T3A by auto
    }
    then have x⁻¹ ∈ ⋂ℋ using assms(1) by auto
  }
  then have ∀x ∈ ⋂ℋ. x⁻¹ ∈ ⋂ℋ by simp
  ultimately show thesis using group0_3_T3 by auto
qed
```

The range of the subgroup operation is the whole subgroup.

```
lemma image_subgr_op: assumes A1: IsAsubgroup(H,P)
  shows restrict(P,H×H)(H×H) = H
proof -
  from A1 have monoid0(H,restrict(P,H×H))
    using IsAsubgroup_def IsAgroup_def monoid0_def
    by simp
  then show thesis by (rule monoid0.range_carr)
qed
```

If we restrict the inverse to a subgroup, then the restricted inverse is onto the subgroup.

```
lemma (in group0) restr_inv_onto: assumes A1: IsAsubgroup(H,P)
```

```
  shows restrict(GroupInv(G,P),H)(H) = H
proof -
  from A1 have GroupInv(H,restrict(P,H×H))(H) = H
    using IsAsubgroup_def group0_def group0.group_inv_surj
    by simp
  with A1 show thesis using group0_3_T1 by simp
qed
```

A union of two subgroups is a subgroup iff one of the subgroups is a subset of the other subgroup.

```
lemma (in group0) union_subgroups:
  assumes IsAsubgroup(H₁,P) and IsAsubgroup(H₂,P)
  shows IsAsubgroup(H₁∪H₂,P) ⟷ (H₁⊆H₂ ∨ H₂⊆H₁)
proof
  assume H₁⊆H₂ ∨ H₂⊆H₁ show IsAsubgroup(H₁∪H₂,P)
  proof -
    from ‹H₁⊆H₂ ∨ H₂⊆H₁› have H₂ = H₁∪H₂ ∨ H₁ = H₁∪H₂ by auto
    with assms show IsAsubgroup(H₁∪H₂,P) by auto
  qed
next
  assume IsAsubgroup(H₁∪H₂, P) show H₁⊆H₂ ∨ H₂⊆H₁
  proof -
    { assume ¬ H₁⊆H₂
      then obtain x where x∈H₁ and x∉H₂ by auto
      with assms(1) have x⁻¹ ∈ H₁ using group0_3_T3A by simp
      { fix y assume y∈H₂
        let z = x·y
        from ‹x∈H₁› ‹y∈H₂› have x ∈ H₁∪H₂ and y ∈ H₁∪H₂ by auto
        with ‹IsAsubgroup(H₁∪H₂,P)› have z ∈ H₁∪H₂ using group0_3_L6
by blast
        from assms ‹x ∈ H₁∪H₂›  ‹y∈H₂› have x∈G y∈G and y⁻¹∈H₂
          using group0_3_T3A group0_3_L2 by auto
        then have z·y⁻¹ = x and x⁻¹·z = y using inv_cancel_two(2,3) by
auto
        { assume z ∈ H₂
          with ‹IsAsubgroup(H₂,P)› ‹y⁻¹∈H₂› have z·y⁻¹ ∈ H₂ using group0_3_L6
by simp
          with ‹z·y⁻¹ = x› ‹x∉H₂› have False by auto
        } hence z ∉ H₂ by auto
        with assms(1) ‹x⁻¹ ∈ H₁› ‹z ∈ H₁∪H₂› have x⁻¹·z ∈ H₁ using group0_3_L6
by simp
        with ‹x⁻¹·z = y› have y∈H₁ by simp
      } hence H₂⊆H₁ by blast
    } thus thesis by blast
  qed
qed
```

Transitivity for ”is a subgroup of” relation. The proof (probably) uses the lemma `restrict_restrict` from standard Isabelle/ZF library which states that restrict(restrict(f,A),B) = restrict(f,A∩B). That lemma is added to the simplifier, so it does not have to be referenced explicitly in the proof below.

```
lemma subgroup_transitive:
  assumes IsAgroup(G₃,P) IsAsubgroup(G₂,P) IsAsubgroup(G₁,restrict(P,G₂×G₂))
  shows IsAsubgroup(G₁,P)
proof -
  from assms(2) have group0(G₂,restrict(P,G₂×G₂)) unfolding IsAsubgroup_def
group0_def by simp
  with assms(3) have G₁⊆G₂ using group0.group0_3_L2 by simp
  hence G₂×G₂ ∩ G₁×G₁ = G₁×G₁ by auto
  with assms(3) show IsAsubgroup(G₁,P) unfolding IsAsubgroup_def by simp
qed
```

33.3 Groups vs. loops

We defined groups as monoids with the inverse operation. An alternative way of defining a group is as a loop whose operation is associative.

Groups have left and right division.

```
lemma (in group0) gr_has_lr_div: shows HasLeftDiv(G,P) and HasRightDiv(G,P)
proof -
  { fix x y assume x∈G y∈G
    then have x⁻¹·y ∈ G ∧ x·(x⁻¹·y) = y using group_op_closed inverse_in_group
inv_cancel_two(4)
      by simp
    hence ∃z. z∈G ∧ x·z =y by auto
    moreover
    { fix z₁ z₂ assume z₁∈G ∧ x·z₁ =y and z₂∈G ∧ x·z₂ =y
      with ‹x∈G› have z₁ = z₂ using cancel_left by blast
    }
    ultimately have ∃!z. z∈G ∧ x·z =y by auto
  } then show HasLeftDiv(G,P) unfolding HasLeftDiv_def by simp
  { fix x y assume x∈G y∈G
    then have y·x⁻¹ ∈ G ∧ (y·x⁻¹)·x = y using group_op_closed inverse_in_group
inv_cancel_two(1)
      by simp
    hence ∃z. z∈G ∧ z·x =y by auto
    moreover
    { fix z₁ z₂ assume z₁∈G ∧ z₁·x =y and z₂∈G ∧ z₂·x =y
      with ‹x∈G› have z₁ = z₂ using cancel_right by blast
    }
    ultimately have ∃!z. z∈G ∧ z·x =y by auto
  } then show HasRightDiv(G,P) unfolding HasRightDiv_def by simp
qed
```

A group is a quasigroup and a loop.

```
lemma (in group0) group_is_loop: shows IsAquasigroup(G,P) and IsAloop(G,P)
```

```
proof -
  show IsAquasigroup(G,P) unfolding IsAquasigroup_def HasLatinSquareProp_def
    using gr_has_lr_div group_oper_fun by simp
  then show IsAloop(G,P) unfolding IsAloop_def using group0_2_L2 by auto
qed
```

An associative loop is a group.

```
theorem assoc_loop_is_gr: assumes IsAloop(G,P) and P {is associative
on} G
  shows IsAgroup(G,P)
proof -
  from assms(1) have ∃e∈G. ∀x∈G. P⟨e,x⟩ = x ∧ P⟨x,e⟩ = x
    unfolding IsAloop_def by simp
  with assms(2) have IsAmonoid(G,P) unfolding IsAmonoid_def by simp
  { fix x assume x∈G
    let y = RightInv(G,P)(x)
    from assms(1) ‹x∈G› have y ∈ G and P⟨x,y⟩ = TheNeutralElement(G,P)
      using loop_loop0_valid loop0.lr_inv_props(3,4) by auto
    hence ∃y∈G. P⟨x,y⟩ = TheNeutralElement(G,P) by auto
  }
  with ‹IsAmonoid(G,P)› show IsAgroup(G,P) unfolding IsAgroup_def by
simp
qed
```

For groups the left and right inverse are the same as the group inverse.

```
lemma (in group0) lr_inv_gr_inv:
  shows LeftInv(G,P) = GroupInv(G,P) and RightInv(G,P) = GroupInv(G,P)
proof -
  have LeftInv(G,P):G→G using group_is_loop loop_loop0_valid loop0.lr_inv_fun(1)
    by simp
  moreover from groupAssum have GroupInv(G,P):G→G using group0_2_T2
by simp
  moreover
  { fix x assume x∈G
    let y = LeftInv(G,P)(x)
    from ‹x∈G› have y ∈ G and y·x = 1
      using group_is_loop(2) loop_loop0_valid loop0.lr_inv_props(1,2)
by auto
    with ‹x∈G› have LeftInv(G,P)(x) = GroupInv(G,P)(x) using group0_2_L9(1)
by simp
  }
  ultimately show LeftInv(G,P) = GroupInv(G,P) using func_eq by blast
  have RightInv(G,P):G→G using group_is_loop loop_loop0_valid loop0.lr_inv_fun(2)
    by simp
  moreover from groupAssum have GroupInv(G,P):G→G using group0_2_T2
by simp
  moreover
  { fix x assume x∈G
    let y = RightInv(G,P)(x)
```

```
    from ‹x∈G› have y ∈ G and x·y = 1
      using group_is_loop(2) loop_loop0_valid loop0.lr_inv_props(3,4)
by auto
    with ‹x∈G› have RightInv(G,P)(x) = GroupInv(G,P)(x) using group0_2_L9(2)
by simp
  }
  ultimately show RightInv(G,P) = GroupInv(G,P) using func_eq by blast

qed
```

33.4 Product of a list of group elements

The `group0` context defines notation for a product of a finite list of group elements. This sections shows basic properties of that notation.

The assumptions of the `semigr0` context hold in the `group0` context.

```
lemma (in group0) semigr0_valid_in_group0: shows semigr0(G,P)
  using groupAssum IsAgroup_def IsAmonoid_def semigr0_def by simp
```

Since semigroups do not have a neutral element the product operator in the `semigr0` is defined for nonempty lists only as `Fold1(f,a)`, where f is the semigroup operation and a is a (nonempty) list. This is a bit different from the product of a finite list in the `group0` context. The next lemma helps in translating between these two notations by asserting that in the `group0` context the sum of a finite list s is the same as `Fold1(s1)` where s_1 is the list s with the neutral element of the group prepended to it.

```
lemma (in group0) list_prod_as_fold1: assumes n∈nat s:n→G
  shows (∏s) = Fold1(P,Prepend(s,1))
  using assms prepend_props group0_2_L2 tail_prepend
  unfolding Fold1_def by simp
```

For nonempty lists the product is the the same as `Fold1` of the list with respect to the operation.

```
lemma (in group0) nempty_list_prod_as_fold1: assumes n∈nat s:(n #+ 1)→G
  shows (∏s) = Fold1(P,s)
  using assms group_oper_fun group0_2_L2 fold_detach_first
    succ_add_one(6) apply_funtype
  unfolding Fold1_def by simp
```

The product of a list is an element of the group.

```
lemma (in group0) list_prod_in_group: assumes n∈nat s:n→G
  shows (∏s) ∈ G
proof -
  have P:G×G→G 1∈G G≠∅ using group_oper_fun group0_2_L2 by auto
  with assms show (∏s) ∈ G using fold_props(2) by simp
qed
```

Product of a singleton list is its only element.

```
lemma (in group0) prod_singleton: assumes  s:1→G
  shows (∏s) = s(0)
  using assms nempty_list_prod_as_fold1 semigr0_valid_in_group0 semigr0.prod_of_1elem
  by force
```

The group0 locale defines a notation for a natural power of a group element. The next lemma provides two alternative definitions for that notation: a natural power of a group element x is the product of the constant list $n\{x\}$, i.e. the fold of the group operation starting from the neutral element of $n\{x\}$. It's really the monoid_nat_mult_def_alt lemma from Monoid_ZF_1 theory, just written in the multiplicative notation used in the group0 context.

```
lemma (in group0) group_nat_pow_def_alt:
  shows pow(n,x) = ∏n×{x} and pow(n,x) =  Fold(P,1,n×{x})
  using monoid.monoid_nat_mult_def_alt by simp_all
```

A natural power of an element of the group is an element of the group. This is really lemma nat_mult_type from Monoid_ZF_1 theory written in multiplicative notation.

```
lemma (in group0) nat_pow_type: assumes n∈nat x∈G
  shows pow(n,x) ∈ G using assms monoid.nat_mult_type by simp
```

x raised to the power $n+1$ can be written as $x \cdot x^n$ or $(x^n) \cdot x$. This is just lemma nat_mult_add_one from Monoid_ZF_1 theory written in multiplicative notation.

```
lemma (in group0) nat_pow_add_one: assumes n∈nat x∈G
  shows pow(n #+ 1,x) = pow(n,x)·x and pow(n #+ 1,x) = x·pow(n,x)
  using assms monoid.nat_mult_add_one by simp_all
```

x raised to the power $n+m$ is the product of powers x^n and x^m. This is really lemma nat_mult_add from Monoid_ZF_1 theory written in multiplicative notation.

```
lemma (in group0) nat_pow_sum_exps: assumes n∈nat m∈nat x∈G
  shows pow(n #+ m,x) = pow(n,x)·pow(m,x)
  using assms monoid.nat_mult_add by simp
```

end

34 Groups 1

theory Group_ZF_1 **imports** Group_ZF

begin

In this theory we consider right and left translations and odd functions.

34.1 Translations

In this section we consider translations. Translations are maps $T : G \to G$ of the form $T_g(a) = g \cdot a$ or $T_g(a) = a \cdot g$. We also consider two-dimensional translations $T_g : G \times G \to G \times G$, where $T_g(a, b) = (a \cdot g, b \cdot g)$ or $T_g(a, b) = (g \cdot a, g \cdot b)$.

For an element $a \in G$ the right translation is defined a function (set of pairs) such that its value (the second element of a pair) is the value of the group operation on the first element of the pair and g. This looks a bit strange in the raw set notation, when we write a function explicitely as a set of pairs and value of the group operation on the pair $\langle a, b\rangle$ as `P⟨a,b⟩` instead of the usual infix $a \cdot b$ or $a + b$.

```
definition
  RightTranslation(G,P,g) ≡ {⟨ a,b⟩ ∈ G×G. P⟨a,g⟩ = b}
```

A similar definition of the left translation.

```
definition
  LeftTranslation(G,P,g) ≡ {⟨a,b⟩ ∈ G×G. P⟨g,a⟩ = b}
```

Translations map G into G. Two dimensional translations map $G \times G$ into itself.

```
lemma (in group0) group0_5_L1: assumes A1: g∈G
  shows RightTranslation(G,P,g) : G→G and  LeftTranslation(G,P,g) :
G→G
proof -
  from A1 have ∀a∈G. a·g ∈ G and ∀a∈G. g·a ∈ G
    using group_oper_fun apply_funtype by auto
  then show
    RightTranslation(G,P,g) : G→G
    LeftTranslation(G,P,g) : G→G
    using RightTranslation_def LeftTranslation_def func1_1_L11A
    by auto
qed
```

The values of the translations are what we expect.

```
lemma (in group0) group0_5_L2: assumes g∈G a∈G
  shows
  RightTranslation(G,P,g)(a) = a·g
  LeftTranslation(G,P,g)(a) = g·a
  using assms group0_5_L1 RightTranslation_def LeftTranslation_def
    func1_1_L11B by auto
```

Composition of left translations is a left translation by the product.

```
lemma (in group0) group0_5_L4: assumes A1: g∈G h∈G a∈G and
  A2: T_g = LeftTranslation(G,P,g)  T_h = LeftTranslation(G,P,h)
  shows
```

```
  Tg(Th(a)) = g·h·a
  Tg(Th(a)) = LeftTranslation(G,P,g·h)(a)
proof -
  from A1 have I: h·a∈G  g·h∈G
    using group_oper_fun apply_funtype by auto
  with A1 A2 show Tg(Th(a)) = g·h·a
    using group0_5_L2 group_oper_assoc by simp
  with A1 A2 I show
    Tg(Th(a)) = LeftTranslation(G,P,g·h)(a)
    using group0_5_L2 group_oper_assoc by simp
qed
```

Composition of right translations is a right translation by the product.

```
lemma (in group0) group0_5_L5: assumes A1: g∈G h∈G a∈G and
  A2: Tg = RightTranslation(G,P,g)  Th = RightTranslation(G,P,h)
  shows
 Tg(Th(a)) = a·h·g
  Tg(Th(a)) = RightTranslation(G,P,h·g)(a)
proof -
  from A1 have I: a·h∈G h·g ∈G
    using group_oper_fun apply_funtype by auto
  with A1 A2 show Tg(Th(a)) = a·h·g
    using group0_5_L2 group_oper_assoc by simp
  with A1 A2 I show
    Tg(Th(a)) = RightTranslation(G,P,h·g)(a)
    using group0_5_L2 group_oper_assoc by simp
qed
```

Point free version of group0_5_L4 and group0_5_L5.

```
lemma (in group0) trans_comp: assumes g∈G h∈G shows
  RightTranslation(G,P,g) O RightTranslation(G,P,h) = RightTranslation(G,P,h·g)
  LeftTranslation(G,P,g) O LeftTranslation(G,P,h) = LeftTranslation(G,P,g·h)
proof -
  let Tg = RightTranslation(G,P,g)
  let Th = RightTranslation(G,P,h)
  from assms have Tg:G→G and Th:G→G
    using group0_5_L1 by auto
  then have Tg O Th:G→G using comp_fun by simp
  moreover from assms have RightTranslation(G,P,h·g):G→G
    using group_op_closed group0_5_L1 by simp
  moreover from assms ‹Th:G→G› have
    ∀a∈G. (Tg O Th)(a) = RightTranslation(G,P,h·g)(a)
    using comp_fun_apply group0_5_L5 by simp
  ultimately show Tg O Th =  RightTranslation(G,P,h·g)
    by (rule func_eq)
next
  let Tg = LeftTranslation(G,P,g)
  let Th = LeftTranslation(G,P,h)
  from assms have Tg:G→G and Th:G→G
```

```
    using group0_5_L1 by auto
  then have Tg O Th:G→G using comp_fun by simp
  moreover from assms have LeftTranslation(G,P,g·h):G→G
    using group_op_closed group0_5_L1 by simp
  moreover from assms ‹Th:G→G› have
    ∀a∈G. (Tg O Th)(a) = LeftTranslation(G,P,g·h)(a)
    using comp_fun_apply group0_5_L4 by simp
  ultimately show Tg O Th =  LeftTranslation(G,P,g·h)
    by (rule func_eq)
qed
```

The image of a set under a composition of translations is the same as the image under translation by a product.

```
lemma (in group0) trans_comp_image: assumes A1: g∈G h∈G and
  A2: Tg = LeftTranslation(G,P,g)  Th = LeftTranslation(G,P,h)
shows Tg(Th(A)) = LeftTranslation(G,P,g·h)(A)
proof -
  from A2 have Tg(Th(A)) = (Tg O Th)(A)
    using image_comp by simp
  with assms show thesis using trans_comp by simp
qed
```

Another form of the image of a set under a composition of translations

```
lemma (in group0) group0_5_L6:
  assumes A1: g∈G h∈G and A2: A⊆G and
  A3: Tg = RightTranslation(G,P,g)  Th = RightTranslation(G,P,h)
  shows Tg(Th(A)) = {a·h·g. a∈A}
proof -
  from A2 have ∀a∈A. a∈G by auto
  from A1 A3 have Tg : G→G  Th : G→G
    using group0_5_L1 by auto
  with assms ‹∀a∈A. a∈G› show
    Tg(Th(A)) = {a·h·g. a∈A}
    using func1_1_L15C group0_5_L5 by auto
qed
```

The translation by neutral element is the identity on group.

```
lemma (in group0) trans_neutral: shows
  RightTranslation(G,P,1) = id(G) and LeftTranslation(G,P,1) = id(G)
proof -
  have RightTranslation(G,P,1):G→G and ∀a∈G. RightTranslation(G,P,1)(a)
= a
    using group0_2_L2 group0_5_L1 group0_5_L2  by auto
  then show RightTranslation(G,P,1) = id(G) by (rule indentity_fun)
  have LeftTranslation(G,P,1):G→G and ∀a∈G. LeftTranslation(G,P,1)(a)
= a
    using group0_2_L2 group0_5_L1 group0_5_L2  by auto
  then show LeftTranslation(G,P,1) = id(G) by (rule indentity_fun)
qed
```

Translation by neutral element does not move sets.

```
lemma (in group0) trans_neutral_image: assumes V⊆G
  shows RightTranslation(G,P,1)(V) = V and LeftTranslation(G,P,1)(V)
= V
  using assms trans_neutral image_id_same by auto
```

Composition of translations by an element and its inverse is identity.

```
lemma (in group0) trans_comp_id: assumes g∈G shows
  RightTranslation(G,P,g) O RightTranslation(G,P,g⁻¹) = id(G) and
  RightTranslation(G,P,g⁻¹) O RightTranslation(G,P,g) = id(G) and
  LeftTranslation(G,P,g) O LeftTranslation(G,P,g⁻¹) = id(G) and
  LeftTranslation(G,P,g⁻¹) O LeftTranslation(G,P,g) = id(G)
  using assms inverse_in_group trans_comp group0_2_L6 trans_neutral by
auto
```

Translations are bijective.

```
lemma (in group0) trans_bij: assumes g∈G shows
  RightTranslation(G,P,g) ∈ bij(G,G) and LeftTranslation(G,P,g) ∈ bij(G,G)
proof-
  from assms have
    RightTranslation(G,P,g):G→G and
    RightTranslation(G,P,g⁻¹):G→G and
    RightTranslation(G,P,g) O RightTranslation(G,P,g⁻¹) = id(G)
    RightTranslation(G,P,g⁻¹) O RightTranslation(G,P,g) = id(G)
  using inverse_in_group group0_5_L1 trans_comp_id by auto
  then show RightTranslation(G,P,g) ∈ bij(G,G) using fg_imp_bijective
by simp
  from assms have
    LeftTranslation(G,P,g):G→G and
    LeftTranslation(G,P,g⁻¹):G→G and
    LeftTranslation(G,P,g) O LeftTranslation(G,P,g⁻¹) = id(G)
    LeftTranslation(G,P,g⁻¹) O LeftTranslation(G,P,g) = id(G)
    using inverse_in_group group0_5_L1 trans_comp_id by auto
  then show LeftTranslation(G,P,g) ∈ bij(G,G) using fg_imp_bijective
by simp
qed
```

Converse of a translation is translation by the inverse.

```
lemma (in group0) trans_conv_inv: assumes g∈G shows
  converse(RightTranslation(G,P,g)) = RightTranslation(G,P,g⁻¹) and
  converse(LeftTranslation(G,P,g)) = LeftTranslation(G,P,g⁻¹) and
  LeftTranslation(G,P,g) = converse(LeftTranslation(G,P,g⁻¹)) and
  RightTranslation(G,P,g) = converse(RightTranslation(G,P,g⁻¹))
proof -
  from assms have
    RightTranslation(G,P,g) ∈ bij(G,G)  RightTranslation(G,P,g⁻¹) ∈ bij(G,G)
and
    LeftTranslation(G,P,g) ∈ bij(G,G)  LeftTranslation(G,P,g⁻¹) ∈ bij(G,G)
```

```
    using trans_bij inverse_in_group by auto
  moreover from assms have
    RightTranslation(G,P,g⁻¹) O RightTranslation(G,P,g) = id(G) and
    LeftTranslation(G,P,g⁻¹) O LeftTranslation(G,P,g) = id(G) and
    LeftTranslation(G,P,g) O LeftTranslation(G,P,g⁻¹) = id(G) and
    LeftTranslation(G,P,g⁻¹) O LeftTranslation(G,P,g) = id(G)
    using trans_comp_id by auto
  ultimately show
    converse(RightTranslation(G,P,g)) = RightTranslation(G,P,g⁻¹) and
    converse(LeftTranslation(G,P,g)) = LeftTranslation(G,P,g⁻¹) and
    LeftTranslation(G,P,g) = converse(LeftTranslation(G,P,g⁻¹)) and
    RightTranslation(G,P,g) = converse(RightTranslation(G,P,g⁻¹))
    using comp_id_conv by auto
qed
```

The image of a set by translation is the same as the inverse image by by the inverse element translation.

```
lemma (in group0) trans_image_vimage: assumes g∈G shows
  LeftTranslation(G,P,g)(A) = LeftTranslation(G,P,g⁻¹)-(A) and
  RightTranslation(G,P,g)(A) = RightTranslation(G,P,g⁻¹)-(A)
  using assms trans_conv_inv vimage_converse by auto
```

Another way of looking at translations is that they are sections of the group operation.

```
lemma (in group0) trans_eq_section: assumes g∈G shows
  RightTranslation(G,P,g) = Fix2ndVar(P,g) and
  LeftTranslation(G,P,g) =  Fix1stVar(P,g)
proof -
  let T = RightTranslation(G,P,g)
  let F = Fix2ndVar(P,g)
  from assms have T: G→G and F: G→G
    using group0_5_L1 group_oper_fun fix_2nd_var_fun by auto
  moreover from assms have ∀a∈G. T(a) = F(a)
    using group0_5_L2 group_oper_fun fix_var_val by simp
  ultimately show T = F by (rule func_eq)
next
  let T = LeftTranslation(G,P,g)
  let F = Fix1stVar(P,g)
  from assms have T: G→G and F: G→G
    using group0_5_L1 group_oper_fun fix_1st_var_fun by auto
  moreover from assms have ∀a∈G. T(a) = F(a)
    using group0_5_L2 group_oper_fun fix_var_val by simp
  ultimately show T = F by (rule func_eq)
qed
```

A lemma demonstrating what is the left translation of a set

```
lemma (in group0) ltrans_image: assumes A1: V⊆G and A2: x∈G
  shows LeftTranslation(G,P,x)(V) = {x·v. v∈V}
```

```
proof -
  from assms have LeftTranslation(G,P,x)(V) = {LeftTranslation(G,P,x)(v).
v∈V}
    using group0_5_L1 func_imagedef by blast
  moreover from assms have ∀v∈V. LeftTranslation(G,P,x)(v) = x·v
    using group0_5_L2 by auto
  ultimately show thesis by auto
qed
```

A lemma demonstrating what is the right translation of a set

```
lemma (in group0) rtrans_image: assumes A1: V⊆G and A2: x∈G
  shows RightTranslation(G,P,x)(V) = {v·x. v∈V}
proof -
  from assms have RightTranslation(G,P,x)(V) = {RightTranslation(G,P,x)(v).
v∈V}
    using group0_5_L1 func_imagedef by blast
  moreover from assms have ∀v∈V. RightTranslation(G,P,x)(v) = v·x
    using group0_5_L2 by auto
  ultimately show thesis by auto
qed
```

Right and left translations of a set are subsets of the group. Interestingly, we do not have to assume the set is a subset of the group.

```
lemma (in group0) lrtrans_in_group: assumes x∈G
  shows LeftTranslation(G,P,x)(V) ⊆ G and RightTranslation(G,P,x)(V)
⊆ G
proof -
  from assms have LeftTranslation(G,P,x):G→G and RightTranslation(G,P,x):G→G
    using group0_5_L1 by auto
  then show LeftTranslation(G,P,x)(V) ⊆ G and RightTranslation(G,P,x)(V)
⊆ G
    using func1_1_L6(2) by auto
qed
```

A technical lemma about solving equations with translations.

```
lemma (in group0) ltrans_inv_in: assumes A1: V⊆G and A2: y∈G and
  A3: x ∈ LeftTranslation(G,P,y)(GroupInv(G,P)(V))
  shows y ∈ LeftTranslation(G,P,x)(V)
proof -
  have x∈G
  proof -
    from A2 have LeftTranslation(G,P,y):G→G using group0_5_L1 by simp
    then have LeftTranslation(G,P,y)(GroupInv(G,P)(V)) ⊆ G
      using func1_1_L6 by simp
    with A3 show x∈G by auto
  qed
  have ∃v∈V. x = y·v⁻¹
  proof -
    have GroupInv(G,P): G→G using groupAssum group0_2_T2
```

```
      by simp
    with assms obtain z where z ∈ GroupInv(G,P)(V) and x = y·z
      using func1_1_L6 ltrans_image by auto
    with A1 ‹GroupInv(G,P): G→G› show thesis using func_imagedef by
auto
  qed
  then obtain v where v∈V and x = y·v⁻¹ by auto
  with A1 A2 have y = x·v using inv_cancel_two by auto
  with assms ‹x∈G› ‹v∈V› show thesis using ltrans_image by auto
qed
```

We can look at the result of interval arithmetic operation as union of left translated sets.

```
lemma (in group0) image_ltrans_union: assumes A⊆G B⊆G shows
  (P {lifted to subsets of} G)⟨A,B⟩ = (⋃a∈A.  LeftTranslation(G,P,a)(B))
proof
  from assms have I: (P {lifted to subsets of} G)⟨A,B⟩ = {a·b . ⟨a,b⟩ ∈
A×B}
    using group_oper_fun lift_subsets_explained by simp
  { fix c assume c ∈ (P {lifted to subsets of} G)⟨A,B⟩
    with I obtain a b where c = a·b and a∈A  b∈B by auto
    hence c ∈ {a·b. b∈B} by auto
    moreover from assms ‹a∈A› have
      LeftTranslation(G,P,a)(B) = {a·b. b∈B} using ltrans_image by auto
    ultimately have c ∈ LeftTranslation(G,P,a)(B) by simp
    with ‹a∈A› have c ∈ (⋃a∈A.  LeftTranslation(G,P,a)(B)) by auto
  } thus (P {lifted to subsets of} G)⟨A,B⟩ ⊆ (⋃a∈A.  LeftTranslation(G,P,a)(B))
    by auto
  { fix c assume c ∈ (⋃a∈A.  LeftTranslation(G,P,a)(B))
    then obtain a where a∈A and c ∈ LeftTranslation(G,P,a)(B)
      by auto
    moreover from assms ‹a∈A› have LeftTranslation(G,P,a)(B) = {a·b.
b∈B}
      using ltrans_image by auto
    ultimately obtain b where b∈B and c = a·b by auto
    with I ‹a∈A› have c ∈ (P {lifted to subsets of} G)⟨A,B⟩ by auto
  } thus (⋃a∈A. LeftTranslation(G,P,a)(B)) ⊆ (P {lifted to subsets of}
G)⟨A,B⟩
    by auto
qed
```

The right translation version of `image_ltrans_union` The proof follows the same schema.

```
lemma (in group0) image_rtrans_union: assumes A⊆G B⊆G shows
  (P {lifted to subsets of} G)⟨A,B⟩ = (⋃b∈B.  RightTranslation(G,P,b)(A))
proof
  from assms have I: (P {lifted to subsets of} G)⟨A,B⟩ = {a·b . ⟨a,b⟩ ∈
A×B}
    using group_oper_fun lift_subsets_explained by simp
```

```
  { fix c assume c ∈ (P {lifted to subsets of} G)⟨A,B⟩
    with I obtain a b where c = a·b and a∈A  b∈B by auto
    hence c ∈ {a·b. a∈A} by auto
    moreover from assms ‹b∈B› have
      RightTranslation(G,P,b)(A) = {a·b. a∈A} using rtrans_image by auto
    ultimately have c ∈ RightTranslation(G,P,b)(A) by simp
    with ‹b∈B› have c ∈ (⋃b∈B.  RightTranslation(G,P,b)(A)) by auto
  } thus (P {lifted to subsets of} G)⟨A,B⟩ ⊆ (⋃b∈B.  RightTranslation(G,P,b)(A))
    by auto
  { fix c assume c ∈ (⋃b∈B.  RightTranslation(G,P,b)(A))
    then obtain b where b∈B and c ∈ RightTranslation(G,P,b)(A)
      by auto
    moreover from assms ‹b∈B› have RightTranslation(G,P,b)(A) = {a·b.
a∈A}
      using rtrans_image by auto
    ultimately obtain a where a∈A and c = a·b by auto
    with I ‹b∈B› have c ∈ (P {lifted to subsets of} G)⟨A,B⟩ by auto
  } thus (⋃b∈B. RightTranslation(G,P,b)(A)) ⊆ (P {lifted to subsets
of} G)⟨A,B⟩
    by auto
qed
```

If the neutral element belongs to a set, then an element of group belongs the translation of that set.

```
lemma (in group0) neut_trans_elem:
  assumes A1: A⊆G g∈G and A2: 1∈A
  shows g ∈ LeftTranslation(G,P,g)(A) g ∈ RightTranslation(G,P,g)(A)
proof -
  from assms have g·1 ∈ LeftTranslation(G,P,g)(A)
    using ltrans_image by auto
  with A1 show g ∈ LeftTranslation(G,P,g)(A) using group0_2_L2 by simp
  from assms have 1·g ∈ RightTranslation(G,P,g)(A)
    using rtrans_image by auto
  with A1 show g ∈ RightTranslation(G,P,g)(A) using group0_2_L2 by simp
qed
```

The neutral element belongs to the translation of a set by the inverse of an element that belongs to it.

```
lemma (in group0) elem_trans_neut: assumes A1: A⊆G and A2: g∈A
  shows 1 ∈ LeftTranslation(G,P,g⁻¹)(A) 1 ∈ RightTranslation(G,P,g⁻¹)(A)
proof -
  from assms have ginv:g⁻¹ ∈ G using inverse_in_group by auto
  with assms have g⁻¹·g ∈ LeftTranslation(G,P,g⁻¹)(A)
    using ltrans_image by auto
  moreover from assms have g⁻¹·g = 1 using group0_2_L6 by auto
  ultimately show 1 ∈ LeftTranslation(G,P,g⁻¹)(A) by simp
  from ginv assms have g·g⁻¹ ∈ RightTranslation(G,P,g⁻¹)(A)
    using rtrans_image by auto
  moreover from assms have g·g⁻¹ = 1 using group0_2_L6 by auto
```

```
  ultimately show 1 ∈ RightTranslation(G,P,g⁻¹)(A) by simp
qed
```

34.2 Odd functions

This section is about odd functions.

Odd functions are those that commute with the group inverse: $f(a^{-1}) = (f(a))^{-1}$.

```
definition
  IsOdd(G,P,f) ≡ (∀a∈G. f(GroupInv(G,P)(a)) = GroupInv(G,P)(f(a)) )
```

Let's see the definition of an odd function in a more readable notation.

```
lemma (in group0) group0_6_L1:
  shows IsOdd(G,P,p) ⟷ ( ∀a∈G. p(a⁻¹) = (p(a))⁻¹ )
  using IsOdd_def by simp
```

We can express the definition of an odd function in two ways.

```
lemma (in group0) group0_6_L2:
  assumes A1: p : G→G
  shows
  (∀a∈G. p(a⁻¹) = (p(a))⁻¹) ⟷ (∀a∈G. (p(a⁻¹))⁻¹ = p(a))
proof
  assume ∀a∈G. p(a⁻¹) = (p(a))⁻¹
  with A1 show ∀a∈G. (p(a⁻¹))⁻¹ = p(a)
    using apply_funtype group_inv_of_inv by simp
next assume A2: ∀a∈G. (p(a⁻¹))⁻¹ = p(a)
  { fix a assume a∈G
    with A1 A2  have
      p(a⁻¹) ∈ G and ((p(a⁻¹))⁻¹)⁻¹ =  (p(a))⁻¹
    using apply_funtype inverse_in_group by auto
  then have p(a⁻¹) = (p(a))⁻¹
    using group_inv_of_inv by simp
  } then show ∀a∈G. p(a⁻¹) = (p(a))⁻¹ by simp
qed
```

34.3 Subgroups and interval arithmetic

The section `Binary operations` in the `func_ZF` theory defines the notion of "lifting operation to subsets". In short, every binary operation $f : X \times X \longrightarrow X$ on a set X defines an operation on the subsets of X defined by $F(A, B) = \{f\langle x, y\rangle | x \in A, y \in B\}$. In the group context using multiplicative notation we can write this as $H \cdot K = \{x \cdot y | x \in A, y \in B\}$. Similarly we can define $H^{-1} = \{x^{-1} | x \in H\}$. In this section we study properties of these derived operations and how they relate to the concept of subgroups.

The next locale extends the `groups0` locale with notation related to interval arithmetics.

```
locale group4 = group0 +
  fixes sdot (infixl · 70)
  defines sdot_def [simp]: A·B  ≡ (P {lifted to subsets of} G)⟨A,B⟩

  fixes sinv (_⁻¹  [90] 91)
  defines sinv_def[simp]: A⁻¹ ≡ GroupInv(G,P)(A)
```

The next lemma shows a somewhat more explicit way of defining the product of two subsets of a group.

```
lemma (in group4) interval_prod: assumes A⊆G B⊆G
  shows A·B =  {x·y. ⟨x,y⟩ ∈ A×B}
  using assms group_oper_fun lift_subsets_explained by auto
```

Product of elements of subsets of the group is in the set product of those subsets

```
lemma (in group4) interval_prod_el: assumes A⊆G B⊆G x∈A y∈B
  shows x·y ∈ A·B
  using assms interval_prod by auto
```

An alternative definition of a group inverse of a set.

```
lemma (in group4) interval_inv: assumes A⊆G
  shows A⁻¹ = {x⁻¹.x∈A}
proof -
  from groupAssum have GroupInv(G,P):G→G using group0_2_T2 by simp
  with assms show A⁻¹ = {x⁻¹.x∈A} using func_imagedef by simp
qed
```

Group inverse of a set is a subset of the group. Interestingly we don't need to assume the set is a subset of the group.

```
lemma (in group4) interval_inv_cl: shows A⁻¹ ⊆ G
proof -
  from groupAssum have GroupInv(G,P):G→G using group0_2_T2 by simp
  then show A⁻¹ ⊆ G using func1_1_L6(2) by simp
qed
```

The product of two subsets of a group is a subset of the group.

```
lemma (in group4) interval_prod_closed: assumes A⊆G B⊆G
  shows A·B ⊆ G
proof
  fix z assume z ∈ A·B
  with assms obtain x y where x∈A y∈B z=x·y using interval_prod by auto
  with assms show z∈G using group_op_closed by auto
qed
```

The product of sets operation is associative.

```
lemma (in group4) interval_prod_assoc: assumes A⊆G B⊆G C⊆G
  shows A·B·C = A·(B·C)
```

```
proof -
  from groupAssum have (P {lifted to subsets of} G) {is associative on}
Pow(G)
    unfolding IsAgroup_def IsAmonoid_def using lift_subset_assoc by simp
  with assms show thesis unfolding IsAssociative_def by auto
qed
```

A simple rearrangement following from associativity of the product of sets operation.

```
lemma (in group4) interval_prod_rearr1: assumes A⊆G B⊆G C⊆G D⊆G
  shows A·B·(C·D) = A·(B·C)·D
proof -
  from assms(1,2) have A·B ⊆ G using interval_prod_closed by simp
  with assms(3,4) have A·B·(C·D) = A·B·C·D
    using interval_prod_assoc by simp
  also from assms(1,2,3) have A·B·C·D = A·(B·C)·D
    using interval_prod_assoc by simp
  finally show thesis by simp
qed
```

A subset A of the group is closed with respect to the group operation iff $A \cdot A \subseteq A$.

```
lemma (in group4) subset_gr_op_cl: assumes A⊆G
  shows (A {is closed under} P) ⟷ A·A ⊆ A
proof
  assume A {is closed under} P
  { fix z assume z ∈ A·A
    with assms obtain x y where x∈A y∈A and z=x·y using interval_prod
by auto
    with ‹A {is closed under} P› have z∈A unfolding IsOpClosed_def by
simp
  } thus A·A ⊆ A by auto
next
  assume A·A ⊆ A
  { fix x y assume x∈A y∈A
    with assms have x·y ∈ A·A using interval_prod by auto
    with ‹A·A ⊆ A› have x·y ∈ A by auto
  } then show A {is closed under} P unfolding IsOpClosed_def by simp
qed
```

Inverse and square of a subgroup is this subgroup.

```
lemma (in group4) subgroup_inv_sq: assumes IsAsubgroup(H,P)
   shows H⁻¹ = H and H·H = H
proof
  from assms have H⊆G using group0_3_L2 by simp
  with assms show H⁻¹ ⊆ H using interval_inv group0_3_T3A by auto
  { fix x assume x∈H
    with assms have (x⁻¹)⁻¹ ∈ {y⁻¹.y∈H} using group0_3_T3A by auto
```

```
    moreover from ‹x∈H› ‹H⊆G› have (x^{-1})^{-1} = x using group_inv_of_inv
by auto
    ultimately have x ∈ {y^{-1}.y∈H} by auto
    with ‹H⊆G› have x ∈ H^{-1} using interval_inv by simp
  } thus H ⊆ H^{-1} by auto
  from assms have H {is closed under} P using group0_3_L6 unfolding IsOpClosed_def
by simp
  with assms have H·H ⊆ H using subset_gr_op_cl group0_3_L2 by simp
  moreover
  { fix x assume x∈H
    with assms have x∈G using group0_3_L2 by auto
    from assms ‹H⊆G› ‹x∈H› have x·1 ∈ H·H using group0_3_L5 interval_prod
by auto
    with ‹x∈G› have x ∈ H·H using group0_2_L2 by simp
  } hence H ⊆ H·H by auto
  ultimately show H·H = H by auto
qed
```

Inverse of a product two sets is a product of inverses with the reversed order.

```
lemma (in group4) interval_prod_inv: assumes A⊆G B⊆G
  shows
    (A·B)^{-1} = {(x·y)^{-1}.⟨x,y⟩ ∈ A×B}
    (A·B)^{-1} = {y^{-1}·x^{-1}.⟨x,y⟩ ∈ A×B}
    (A·B)^{-1} = (B^{-1})·(A^{-1})
proof -
  from assms have (A·B) ⊆ G using interval_prod_closed by simp
  then have I: (A·B)^{-1} = {z^{-1}.z∈A·B} using interval_inv by simp
  show II: (A·B)^{-1} = {(x·y)^{-1}.⟨x,y⟩ ∈ A×B}
  proof
    { fix p assume p ∈ (A·B)^{-1}
      with I obtain z where p=z^{-1} and z∈A·B by auto
      with assms obtain x y where ⟨x,y⟩ ∈ A×B and z=x·y using interval_prod
by auto
      with ‹p=z^{-1}› have p∈{(x·y)^{-1}.⟨x,y⟩ ∈ A×B} by auto
    } thus (A·B)^{-1} ⊆ {(x·y)^{-1}.⟨x,y⟩ ∈ A×B} by blast
    { fix p assume p∈{(x·y)^{-1}.⟨x,y⟩ ∈ A×B}
      then obtain x y where x∈A y∈B and p=(x·y)^{-1} by auto
      with assms ‹(A·B) ⊆ G› have p∈(A·B)^{-1} using interval_prod interval_inv
        by auto
    } thus {(x·y)^{-1}.⟨x,y⟩ ∈ A×B} ⊆ (A·B)^{-1} by blast
  qed
  have {(x·y)^{-1}.⟨x,y⟩ ∈ A×B} = {y^{-1}·x^{-1}.⟨x,y⟩ ∈ A×B}
  proof
    { fix p assume p ∈ {(x·y)^{-1}.⟨x,y⟩ ∈ A×B}
      then obtain x y where x∈A y∈B and p=(x·y)^{-1} by auto
      with assms have y^{-1}·x^{-1} = (x·y)^{-1} using group_inv_of_two by auto
      with ‹p=(x·y)^{-1}› have p = y^{-1}·x^{-1} by simp
      with ‹x∈A› ‹y∈B› have p∈{y^{-1}·x^{-1}.⟨x,y⟩ ∈ A×B} by auto
    } thus {(x·y)^{-1}.⟨x,y⟩ ∈ A×B} ⊆ {y^{-1}·x^{-1}.⟨x,y⟩ ∈ A×B} by blast
```

```
    { fix p assume p∈{y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B}
      then obtain x y where x∈A y∈B and p=y⁻¹·x⁻¹ by auto
      with assms have p = (x·y)⁻¹ using group_inv_of_two by auto
      with ‹x∈A› ‹y∈B› have p ∈ {(x·y)⁻¹. ⟨x,y⟩ ∈ A×B} by auto
    } thus {y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B} ⊆ {(x·y)⁻¹. ⟨x,y⟩ ∈ A×B} by blast
  qed
  with II show III: (A·B)⁻¹ = {y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B} by simp
  have {y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B} = (B⁻¹)·(A⁻¹)
  proof
    { fix p assume p∈{y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B}
      then obtain x y where x∈A y∈B and p=y⁻¹·x⁻¹ by auto
      with assms have y⁻¹ ∈ (B⁻¹) and x⁻¹ ∈ (A⁻¹)
        using interval_inv by auto
      with ‹p=y⁻¹·x⁻¹› have p ∈ (B⁻¹)·(A⁻¹) using interval_inv_cl interval_prod
        by auto
    } thus {y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B} ⊆ (B⁻¹)·(A⁻¹) by blast
    { fix p assume p ∈ (B⁻¹)·(A⁻¹)
      then obtain y x where y∈B⁻¹ x∈A⁻¹ and p=y·x
        using interval_inv_cl interval_prod by auto
      with assms obtain x₁ y₁ where x₁ ∈ A y₁ ∈ B and x=x₁⁻¹ y=y₁⁻¹ us-
ing interval_inv
        by auto
      with ‹p=y·x› have p ∈ {y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B} by auto
    } thus (B⁻¹)·(A⁻¹) ⊆ {y⁻¹·x⁻¹. ⟨x,y⟩ ∈ A×B} by blast
  qed
  with III show (A·B)⁻¹ = (B⁻¹)·(A⁻¹) by simp
qed
```

If H, K are subgroups then $H \cdot K$ is a subgroup iff $H \cdot K = K \cdot H$.

```
theorem (in group4) prod_subgr_subgr:
  assumes IsAsubgroup(H,P) and IsAsubgroup(K,P)
  shows IsAsubgroup(H·K,P) ⟷  H·K = K·H
proof
  assume IsAsubgroup(H·K,P)
  then have (H·K)⁻¹ = H·K using subgroup_inv_sq(1) by simp
  with assms show H·K = K·H using group0_3_L2 interval_prod_inv subgroup_inv_sq(1)
    by auto
next
  from assms have H⊆G and K⊆G using group0_3_L2 by auto
  have I: H·K ≠ 0
  proof -
    let x = 1 let y = 1
    from assms have x·y ∈ (H·K) using group0_3_L5 group0_3_L2 interval_prod

      by auto
    thus thesis by auto
  qed
  from ‹H⊆G› ‹K⊆G› have II: H·K ⊆ G using interval_prod_closed by simp
```

```
  assume H·K = K·H
  have III: (H·K){is closed under} P
  proof -
    have (H·K)·(H·K) = H·K
    proof -
      from ‹H⊆G› ‹K⊆G› have (H·K)·(H·K) = H·(K·H)·K
        using interval_prod_rearr1 by simp
      also from ‹H·K = K·H› have ... = H·(H·K)·K by simp
      also from ‹H⊆G› ‹K⊆G› have ... = (H·H)·(K·K)
        using interval_prod_rearr1 by simp
      also from assms have ... = H·K using subgroup_inv_sq(2) by simp
      finally show thesis by simp
    qed
    with ‹H·K ⊆ G› show thesis using subset_gr_op_cl by simp
  qed
  have IV: ∀x ∈ H·K. x⁻¹ ∈ H·K
  proof -
    { fix x assume x ∈ H·K
      with ‹H·K ⊆ G› have x⁻¹ ∈ (H·K)⁻¹ using interval_inv by auto
      with assms ‹H⊆G› ‹K⊆G› ‹H·K = K·H› have x⁻¹ ∈ H·K
        using interval_prod_inv subgroup_inv_sq(1) by simp
    } thus thesis by auto
  qed
  from I II III IV show IsAsubgroup(H·K,P) using group0_3_T3 by simp
qed

end
```

35 Groups - an alternative definition

```
theory Group_ZF_1b imports Group_ZF

begin
```

In a typical textbook a group is defined as a set G with an associative operation such that two conditions hold:

A: there is an element $e \in G$ such that for all $g \in G$ we have $e \cdot g = g$ and $g \cdot e = g$. We call this element a "unit" or a "neutral element" of the group.

B: for every $a \in G$ there exists a $b \in G$ such that $a \cdot b = e$, where e is the element of G whose existence is guaranteed by A.

The validity of this definition is rather dubious to me, as condition A does not define any specific element e that can be referred to in condition B - it merely states that a set of such units e is not empty. Of course it does work in the end as we can prove that the set of such neutral elements has exactly one element, but still the definition by itself is not valid. You just can't reference a variable bound by a quantifier outside of the scope of that quantifier.

One way around this is to first use condition A to define the notion of a monoid, then prove the uniqueness of e and then use the condition B to define groups.

Another way is to write conditions A and B together as follows:

$\exists_{e\in G}\ (\forall_{g\in G}\ e\cdot g = g \wedge g\cdot e = g) \wedge (\forall_{a\in G}\exists_{b\in G}\ a\cdot b = e)$.

This is rather ugly.

What I want to talk about is an amusing way to define groups directly without any reference to the neutral elements. Namely, we can define a group as a non-empty set G with an associative operation "$\cdot$" such that

C: for every $a, b \in G$ the equations $a\cdot x = b$ and $y \cdot a = b$ can be solved in G.

This theory file aims at proving the equivalence of this alternative definition with the usual definition of the group, as formulated in `Group_ZF.thy`. The informal proofs come from an Aug. 14, 2005 post by buli on the matematyka.org forum.

35.1 An alternative definition of group

First we will define notation for writing about groups.

We will use the multiplicative notation for the group operation. To do this, we define a context (locale) that tells Isabelle to interpret $a \cdot b$ as the value of function P on the pair $\langle a, b\rangle$.

```
locale group2 =
  fixes P
  fixes dot (infixl · 70)
  defines dot_def [simp]: a · b ≡ P⟨a,b⟩
```

The next theorem states that a set G with an associative operation that satisfies condition C is a group, as defined in IsarMathLib `Group_ZF` theory.

```
theorem (in group2) altgroup_is_group:
  assumes A1: G≠0 and A2: P {is associative on} G
  and A3: ∀a∈G.∀b∈G. ∃x∈G. a·x = b
  and A4: ∀a∈G.∀b∈G. ∃y∈G. y·a = b
  shows IsAgroup(G,P)
proof -
  from A1 obtain a where a∈G by auto
  with A3 obtain x where x∈G and a·x = a
    by auto
  from A4 ‹a∈G› obtain y where y∈G and y·a = a
    by auto
  have I: ∀b∈G. b = b·x ∧ b = y·b
  proof
    fix b assume b∈G
     with A4 ‹a∈G› obtain y_b where y_b∈G
      and y_b·a = b by auto
```

```
    from A3 ‹a∈G› ‹b∈G› obtain x_b where x_b∈G
      and a·x_b = b by auto
    from ‹a·x = a› ‹y·a = a› ‹y_b·a = b› ‹a·x_b = b›
    have b = y_b·(a·x) and b = (y·a)·x_b
      by auto
    moreover from A2 ‹a∈G› ‹x∈G› ‹y∈G› ‹x_b∈G› ‹y_b∈G› have
      (y·a)·x_b = y·(a·x_b)  y_b·(a·x) = (y_b·a)·x
      using IsAssociative_def by auto
    moreover from ‹y_b·a = b› ‹a·x_b = b› have
      (y_b·a)·x = b·x  y·(a·x_b) = y·b
      by auto
    ultimately show b = b·x ∧ b = y·b by simp
  qed
  moreover have x = y
  proof -
    from ‹x∈G› I have x = y·x by simp
    also from ‹y∈G› I have y·x = y by simp
    finally show x = y by simp
  qed
  ultimately have ∀b∈G. b·x = b ∧ x·b = b by simp
  with A2 ‹x∈G› have IsAmonoid(G,P) using IsAmonoid_def by auto
  with A3 show IsAgroup(G,P)
    using monoid0_def monoid0.unit_is_neutral IsAgroup_def
    by simp
qed
```

The converse of altgroup_is_group: in every (classically defined) group condition C holds. In informal mathematics we can say ”Obviously condition C holds in any group.” In formalized mathematics the word ”obviously” is not in the language. The next theorem is proven in the context called group0 defined in the theory Group_ZF.thy. Similarly to the group2 that context defines $a \cdot b$ as $P\langle a, b\rangle$ It also defines notation related to the group inverse and adds an assumption that the pair (G, P) is a group to all its theorems. This is why in the next theorem we don't explicitely assume that (G, P) is a group - this assumption is implicit in the context.

```
theorem (in group0) group_is_altgroup: shows
  ∀a∈G.∀b∈G. ∃x∈G. a·x = b and ∀a∈G.∀b∈G. ∃y∈G. y·a = b
proof -
  { fix a b assume a∈G  b∈G
    let x = a⁻¹· b
    let y = b·a⁻¹
    from ‹a∈G›  ‹b∈G›  have
      x ∈ G  y ∈ G  and  a·x = b  y·a = b
      using inverse_in_group group_op_closed inv_cancel_two
      by auto
    hence ∃x∈G. a·x = b and ∃y∈G. y·a = b by auto
  } thus
      ∀a∈G.∀b∈G. ∃x∈G. a·x = b and
```

```
    ∀a∈G.∀b∈G. ∃y∈G. y·a = b
    by auto
qed
```

An associative quasigroup is a group. This is a bit weaker than `altgroup_is_group` as the definition of quasigroup requires uniqueness of solutions of $a \cdot x = b$ and $y \cdot a = b$ equations.

```
lemma assoc_quasigroup_group:
  assumes IsAquasigroup(G,P) and P {is associative on} G G≠∅
  shows IsAgroup(G,P)
proof -
  from assms(1) have
    ∀a∈G.∀b∈G. ∃y∈G. P⟨a,y⟩ = b and ∀a∈G.∀b∈G. ∃x∈G. P⟨x,a⟩ = b
    unfolding IsAquasigroup_def HasLatinSquareProp_def HasLeftDiv_def
HasRightDiv_def
    using ZF1_1_L9(4) by simp_all
  with assms(2,3) show thesis using group2.altgroup_is_group by auto
qed

end
```

36 Abelian Group

```
theory AbelianGroup_ZF imports Group_ZF

begin
```

A group is called "abelian" if its operation is commutative, i.e. $P\langle a,b\rangle = P\langle a,b\rangle$ for all group elements a, b, where P is the group operation. It is customary to use the additive notation for abelian groups, so this condition is typically written as $a+b = b+a$. We will be using multiplicative notation though (in which the commutativity condition of the operation is written as $a \cdot b = b \cdot a$), just to avoid the hassle of changing the notation we used for general groups.

36.1 Rearrangement formulae

This section is not interesting and should not be read. Here we will prove formulas is which right hand side uses the same factors as the left hand side, just in different order. These facts are obvious in informal math sense, but Isabelle prover is not able to derive them automatically, so we have to provide explicit proofs.

Proving the facts about associative and commutative operations is quite tedious in formalized mathematics. To a human the thing is simple: we can arrange the elements in any order and put parantheses wherever we want, it is all the same. However, formalizing this statement would be rather difficult (I think). The next lemma attempts a quasi-algorithmic approach to this type of problem. To prove that two expressions are equal, we first strip one from parantheses, then rearrange the elements in proper order, then put the parantheses where we want them to be. The algorithm for rearrangement is easy to describe: we keep putting the first element (from the right) that is in the wrong place at the left-most position until we get the proper arrangement. As far removing parantheses is concerned Isabelle does its job automatically.

```
lemma (in group0) group0_4_L2:
  assumes A1:P {is commutative on} G
  and A2:a∈G b∈G c∈G d∈G E∈G F∈G
  shows (a·b)·(c·d)·(E·F) = (a·(d·F))·(b·(c·E))
proof -
  from A2 have (a·b)·(c·d)·(E·F) = a·b·c·d·E·F
    using group_op_closed group_oper_assoc
    by simp
  also have  a·b·c·d·E·F = a·d·F·b·c·E
  proof -
    from A1 A2 have a·b·c·d·E·F = F·(a·b·c·d·E)
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have F·(a·b·c·d·E) = F·a·b·c·d·E
      using group_op_closed group_oper_assoc
      by simp
    also from A1 A2 have F·a·b·c·d·E = d·(F·a·b·c)·E
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have d·(F·a·b·c)·E = d·F·a·b·c·E
      using group_op_closed group_oper_assoc
      by simp
    also from A1 A2 have  d·F·a·b·c·E = a·(d·F)·b·c·E
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have a·(d·F)·b·c·E = a·d·F·b·c·E
      using group_op_closed group_oper_assoc
      by simp
    finally show thesis by simp
  qed
  also from A2 have a·d·F·b·c·E = (a·(d·F))·(b·(c·E))
    using group_op_closed group_oper_assoc
    by simp
  finally show thesis by simp
qed
```

Another useful rearrangement.

```
lemma (in group0) group0_4_L3:
  assumes A1:P {is commutative on} G
```

```
  and A2: a∈G  b∈G and A3: c∈G  d∈G  E∈G  F∈G
  shows a·b·((c·d)^-1·(E·F)^-1) = (a·(E·c)^-1)·(b·(F·d)^-1)
proof -
  from A3 have T1:
    c^-1∈G d^-1∈G E^-1∈G F^-1∈G (c·d)^-1∈G (E·F)^-1∈G
    using inverse_in_group group_op_closed
    by auto
  from A2 T1 have
    a·b·((c·d)^-1·(E·F)^-1) = a·b·(c·d)^-1·(E·F)^-1
    using group_op_closed group_oper_assoc
    by simp
  also from A2 A3 have
    a·b·(c·d)^-1·(E·F)^-1 = (a·b)·(d^-1·c^-1)·(F^-1·E^-1)
    using group_inv_of_two by simp
   also from A1 A2 T1 have
    (a·b)·(d^-1·c^-1)·(F^-1·E^-1) = (a·(c^-1·E^-1))·(b·(d^-1·F^-1))
    using group0_4_L2 by simp
  also from A2 A3 have
    (a·(c^-1·E^-1))·(b·(d^-1·F^-1)) = (a·(E·c)^-1)·(b·(F·d)^-1)
    using group_inv_of_two by simp
  finally show thesis by simp
qed
```

Some useful rearrangements for two elements of a group.

```
lemma (in group0) group0_4_L4:
  assumes A1:P {is commutative on} G
  and A2: a∈G b∈G
  shows
  b^-1·a^-1 = a^-1·b^-1
  (a·b)^-1 = a^-1·b^-1
  (a·b^-1)^-1 = a^-1·b
proof -
  from A2 have T1: b^-1∈G a^-1∈G using inverse_in_group by auto
  with A1 show b^-1·a^-1 = a^-1·b^-1 using IsCommutative_def by simp
  with A2 show (a·b)^-1 = a^-1·b^-1 using group_inv_of_two by simp
  from A2 T1 have (a·b^-1)^-1 = (b^-1)^-1·a^-1 using group_inv_of_two by simp
  with A1 A2 T1 show (a·b^-1)^-1 = a^-1·b
    using group_inv_of_inv IsCommutative_def by simp
qed
```

Another bunch of useful rearrangements with three elements.

```
lemma (in group0) group0_4_L4A:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·c = c·a·b
  a^-1·(b^-1·c^-1)^-1 = (a·(b·c)^-1)^-1
  a·(b·c)^-1 = a·b^-1·c^-1
  a·(b·c^-1)^-1 = a·b^-1·c
```

```
  a·b⁻¹·c⁻¹ = a·c⁻¹·b⁻¹
proof -
  from A1 A2 have a·b·c = c·(a·b)
    using IsCommutative_def group_op_closed
    by simp
  with A2 show a·b·c = c·a·b using
     group_op_closed group_oper_assoc
    by simp
  from A2 have T:
    b⁻¹∈G  c⁻¹∈G  b⁻¹·c⁻¹ ∈ G  a·b ∈ G
    using inverse_in_group group_op_closed
    by auto
  with A1 A2 show a⁻¹·(b⁻¹·c⁻¹)⁻¹ = (a·(b·c)⁻¹)⁻¹
    using group_inv_of_two IsCommutative_def
    by simp
  from A1 A2 T have a·(b·c)⁻¹ = a·(b⁻¹·c⁻¹)
    using group_inv_of_two IsCommutative_def by simp
  with A2 T show a·(b·c)⁻¹ = a·b⁻¹·c⁻¹
    using group_oper_assoc by simp
  from A1 A2 T have a·(b·c⁻¹)⁻¹ = a·(b⁻¹·(c⁻¹)⁻¹)
    using group_inv_of_two IsCommutative_def by simp
  with A2 T show a·(b·c⁻¹)⁻¹ = a·b⁻¹·c
    using group_oper_assoc group_inv_of_inv by simp
  from A1 A2 T have a·b⁻¹·c⁻¹ = a·(c⁻¹·b⁻¹)
    using group_oper_assoc IsCommutative_def by simp
  with A2 T show a·b⁻¹·c⁻¹ = a·c⁻¹·b⁻¹
    using group_oper_assoc by simp
qed
```

Another useful rearrangement.

```
lemma (in group0) group0_4_L4B:
  assumes P {is commutative on} G
  and a∈G  b∈G  c∈G
  shows a·b⁻¹·(b·c⁻¹) = a·c⁻¹
  using assms inverse_in_group group_op_closed
    group0_4_L4 group_oper_assoc inv_cancel_two by simp
```

A couple of permutations of order for three alements.

```
lemma (in group0) group0_4_L4C:
  assumes A1: P {is commutative on} G
  and A2: a∈G b∈G c∈G
  shows
  a·b·c = c·a·b
  a·b·c = a·(c·b)
  a·b·c = c·(a·b)
  a·b·c = c·b·a
proof -
  from A1 A2 show I: a·b·c = c·a·b
    using group0_4_L4A by simp
```

```
  also from A1 A2 have c·a·b = a·c·b
    using IsCommutative_def by simp
  also from A2 have a·c·b = a·(c·b)
    using group_oper_assoc by simp
  finally show a·b·c = a·(c·b) by simp
  from A2 I show a·b·c = c·(a·b)
    using group_oper_assoc by simp
  also from A1 A2 have c·(a·b) = c·(b·a)
    using IsCommutative_def by simp
  also from A2 have c·(b·a) = c·b·a
    using group_oper_assoc by simp
  finally show a·b·c = c·b·a by simp
qed
```

Some rearangement with three elements and inverse.

```
lemma (in group0) group0_4_L4D:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a⁻¹·b⁻¹·c = c·a⁻¹·b⁻¹
  b⁻¹·a⁻¹·c = c·a⁻¹·b⁻¹
  (a⁻¹·b·c)⁻¹ = a·b⁻¹·c⁻¹
proof -
  from A2 have T:
    a⁻¹ ∈ G  b⁻¹ ∈ G  c⁻¹∈G
    using inverse_in_group by auto
  with A1 A2 show
    a⁻¹·b⁻¹·c = c·a⁻¹·b⁻¹
    b⁻¹·a⁻¹·c = c·a⁻¹·b⁻¹
    using  group0_4_L4A by auto
  from A1 A2 T show (a⁻¹·b·c)⁻¹ = a·b⁻¹·c⁻¹
    using group_inv_of_three group_inv_of_inv group0_4_L4C
    by simp
qed
```

Another rearrangement lemma with three elements and equation.

```
lemma (in group0) group0_4_L5: assumes A1:P {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  and A3: c = a·b⁻¹
  shows a = b·c
proof -
  from A2 A3 have c·(b⁻¹)⁻¹ = a
    using inverse_in_group group0_2_L18
    by simp
  with A1 A2 show thesis using
     group_inv_of_inv IsCommutative_def by simp
qed
```

In abelian groups we can cancel an element with its inverse even if separated by another element.

```
lemma (in group0) group0_4_L6A: assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G
  shows
  a·b·a⁻¹ = b
  a⁻¹·b·a = b
  a⁻¹·(b·a) = b
  a·(b·a⁻¹) = b
proof -
  from A1 A2 have
    a·b·a⁻¹ = a⁻¹·a·b
    using inverse_in_group group0_4_L4A by blast
  also from A2 have ... = b
    using group0_2_L6 group0_2_L2 by simp
  finally show a·b·a⁻¹ = b by simp
  from A1 A2 have
    a⁻¹·b·a = a·a⁻¹·b
    using inverse_in_group group0_4_L4A by blast
  also from A2 have ... = b
    using group0_2_L6 group0_2_L2 by simp
  finally show a⁻¹·b·a = b by simp
  moreover from A2 have a⁻¹·b·a = a⁻¹·(b·a)
    using inverse_in_group group_oper_assoc by simp
  ultimately show a⁻¹·(b·a) = b by simp
  from A1 A2 show a·(b·a⁻¹) = b
     using inverse_in_group IsCommutative_def inv_cancel_two
     by simp
qed
```

Another lemma about cancelling with two elements.

```
lemma (in group0) group0_4_L6AA:
  assumes A1: P {is commutative on} G and A2: a∈G  b∈G
  shows a·b⁻¹·a⁻¹ = b⁻¹
  using assms inverse_in_group group0_4_L6A
  by auto
```

Another lemma about cancelling with two elements.

```
lemma (in group0) group0_4_L6AB:
  assumes A1: P {is commutative on} G and A2: a∈G  b∈G
  shows
  a·(a·b)⁻¹ = b⁻¹
  a·(b·a⁻¹) = b
proof -
    from A2 have a·(a·b)⁻¹ = a·(b⁻¹·a⁻¹)
      using group_inv_of_two by simp
    also from A2 have ... = a·b⁻¹·a⁻¹
      using inverse_in_group group_oper_assoc by simp
    also from A1 A2 have ... =  b⁻¹
      using group0_4_L6AA by simp
```

```
    finally show a·(a·b)⁻¹ = b⁻¹ by simp
    from A1 A2 have a·(b·a⁻¹) = a·(a⁻¹·b)
      using inverse_in_group IsCommutative_def by simp
    also from A2 have ... = b
      using inverse_in_group group_oper_assoc group0_2_L6 group0_2_L2
      by simp
    finally show a·(b·a⁻¹) = b by simp
qed
```

Another lemma about cancelling with two elements.

```
lemma (in group0) group0_4_L6AC:
  assumes P {is commutative on} G and a∈G  b∈G
  shows a·(a·b⁻¹)⁻¹ = b
  using assms inverse_in_group group0_4_L6AB group_inv_of_inv
  by simp
```

In abelian groups we can cancel an element with its inverse even if separated by two other elements.

```
lemma (in group0) group0_4_L6B: assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·c·a⁻¹ = b·c
  a⁻¹·b·c·a = b·c
proof -
   from A2 have
     a·b·c·a⁻¹ = a·(b·c)·a⁻¹
     a⁻¹·b·c·a = a⁻¹·(b·c)·a
    using group_op_closed group_oper_assoc inverse_in_group
    by auto
  with A1 A2 show
    a·b·c·a⁻¹ = b·c
    a⁻¹·b·c·a = b·c
    using group_op_closed group0_4_L6A
    by auto
qed
```

In abelian groups we can cancel an element with its inverse even if separated by three other elements.

```
lemma (in group0) group0_4_L6C: assumes A1: P {is commutative on} G
  and A2: a∈G b∈G c∈G d∈G
  shows a·b·c·d·a⁻¹ = b·c·d
proof -
  from A2 have a·b·c·d·a⁻¹ = a·(b·c·d)·a⁻¹
    using group_op_closed group_oper_assoc
    by simp
  with A1 A2 show thesis
    using group_op_closed group0_4_L6A
    by simp
```

qed

Another couple of useful rearrangements of three elements and cancelling.

```
lemma (in group0) group0_4_L6D:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b⁻¹·(a·c⁻¹)⁻¹ = c·b⁻¹
  (a·c)⁻¹·(b·c) = a⁻¹·b
  a·(b·(c·a⁻¹·b⁻¹)) = c
  a·b·c⁻¹·(c·a⁻¹) = b
proof -
  from A2 have T:
    a⁻¹ ∈ G  b⁻¹ ∈ G  c⁻¹ ∈ G
    a·b ∈ G  a·b⁻¹ ∈ G  c⁻¹·a⁻¹ ∈ G  c·a⁻¹ ∈ G
    using inverse_in_group group_op_closed by auto
  with A1 A2 show a·b⁻¹·(a·c⁻¹)⁻¹ = c·b⁻¹
    using group0_2_L12 group_oper_assoc group0_4_L6B
    IsCommutative_def by simp
  from A2 T have (a·c)⁻¹·(b·c) = c⁻¹·a⁻¹·b·c
    using group_inv_of_two group_oper_assoc by simp
  also from A1 A2 T have ... = a⁻¹·b
    using group0_4_L6B by simp
  finally show (a·c)⁻¹·(b·c) = a⁻¹·b
    by simp
  from A1 A2 T show a·(b·(c·a⁻¹·b⁻¹)) = c
    using group_oper_assoc group0_4_L6B group0_4_L6A
    by simp
  from T have a·b·c⁻¹·(c·a⁻¹) = a·b·(c⁻¹·(c·a⁻¹))
    using group_oper_assoc by simp
  also from A1 A2 T have ... = b
    using group_oper_assoc group0_2_L6 group0_2_L2 group0_4_L6A
    by simp
  finally show a·b·c⁻¹·(c·a⁻¹) = b by simp
qed
```

Another useful rearrangement of three elements and cancelling.

```
lemma (in group0) group0_4_L6E:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·(a·c)⁻¹ = b·c⁻¹
proof -
  from A2 have T: b⁻¹ ∈ G  c⁻¹ ∈ G
    using inverse_in_group by auto
  with A1 A2 have
    a·(b⁻¹)⁻¹·(a·(c⁻¹)⁻¹)⁻¹ = c⁻¹·(b⁻¹)⁻¹
    using group0_4_L6D by simp
  with A1 A2 T show a·b·(a·c)⁻¹ = b·c⁻¹
```

```
    using group_inv_of_inv IsCommutative_def
    by simp
qed
```

A rearrangement with two elements and canceelling, special case of `group0_4_L6D` when $c = b^{-1}$.

```
lemma (in group0) group0_4_L6F:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G
  shows a·b⁻¹·(a·b)⁻¹ = b⁻¹·b⁻¹
proof -
  from A2 have b⁻¹ ∈ G
    using inverse_in_group by simp
  with A1 A2 have a·b⁻¹·(a·(b⁻¹)⁻¹)⁻¹ = b⁻¹·b⁻¹
    using group0_4_L6D by simp
  with A2 show a·b⁻¹·(a·b)⁻¹ = b⁻¹·b⁻¹
    using group_inv_of_inv by simp
qed
```

Some other rearrangements with four elements. The algorithm for proof as in `group0_4_L2` works very well here.

```
lemma (in group0) rearr_ab_gr_4_elemA:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b·c·d = a·d·b·c
  a·b·c·d = a·c·(b·d)
proof -
  from A1 A2 have a·b·c·d = d·(a·b·c)
    using  IsCommutative_def group_op_closed
    by simp
  also from A2 have ... = d·a·b·c
    using group_op_closed group_oper_assoc
    by simp
  also from A1 A2 have ... = a·d·b·c
    using IsCommutative_def group_op_closed
    by simp
  finally show a·b·c·d = a·d·b·c
    by simp
  from A1 A2 have a·b·c·d = c·(a·b)·d
    using IsCommutative_def group_op_closed
    by simp
  also from A2 have ... = c·a·b·d
    using group_op_closed group_oper_assoc
    by simp
  also from A1 A2 have ... = a·c·b·d
    using IsCommutative_def group_op_closed
    by simp
  also from A2 have ... = a·c·(b·d)
```

```
    using group_op_closed group_oper_assoc
    by simp
  finally show a·b·c·d = a·c·(b·d)
    by simp
qed
```

Some rearrangements with four elements and inverse that are applications of `rearr_ab_gr_4_elem`

```
lemma (in group0) rearr_ab_gr_4_elemB:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b⁻¹·c⁻¹·d⁻¹ = a·d⁻¹·b⁻¹·c⁻¹
  a·b·c·d⁻¹ = a·d⁻¹·b·c
  a·b·c⁻¹·d⁻¹ =  a·c⁻¹·(b·d⁻¹)
proof -
  from A2 have T: b⁻¹ ∈ G  c⁻¹ ∈ G  d⁻¹ ∈ G
    using inverse_in_group by auto
  with A1 A2 show
    a·b⁻¹·c⁻¹·d⁻¹ = a·d⁻¹·b⁻¹·c⁻¹
    a·b·c·d⁻¹ = a·d⁻¹·b·c
    a·b·c⁻¹·d⁻¹ =  a·c⁻¹·(b·d⁻¹)
    using rearr_ab_gr_4_elemA by auto
qed
```

Some rearrangement lemmas with four elements.

```
lemma (in group0) group0_4_L7:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b·c·d⁻¹ = a·d⁻¹· b·c
  a·d·(b·d·(c·d))⁻¹ = a·(b·c)⁻¹·d⁻¹
  a·(b·c)·d = a·b·d·c
proof -
  from A2 have T:
    b·c ∈ G d⁻¹ ∈ G b⁻¹∈G c⁻¹∈G
    d⁻¹·b ∈ G c⁻¹·d ∈ G (b·c)⁻¹ ∈ G
    b·d ∈ G  b·d·c ∈ G  (b·d·c)⁻¹ ∈ G
    a·d ∈ G  b·c ∈ G
    using group_op_closed inverse_in_group
    by auto
  with A1 A2 have a·b·c·d⁻¹ = a·(d⁻¹·b·c)
    using group_oper_assoc group0_4_L4A by simp
  also from A2 T have a·(d⁻¹·b·c) = a·d⁻¹·b·c
    using group_oper_assoc by simp
  finally show a·b·c·d⁻¹ = a·d⁻¹· b·c by simp
  from A2 T have a·d·(b·d·(c·d))⁻¹ = a·d·(d⁻¹·(b·d·c)⁻¹)
    using group_oper_assoc group_inv_of_two by simp
  also from A2 T have ... = a·(b·d·c)⁻¹
```

```
    using group_oper_assoc inv_cancel_two by simp
  also from A1 A2 have ... =  a·(d·(b·c))⁻¹
    using IsCommutative_def group_oper_assoc by simp
  also from A2 T have ... = a·((b·c)⁻¹·d⁻¹)
    using group_inv_of_two by simp
  also from A2 T have ... =  a·(b·c)⁻¹·d⁻¹
    using group_oper_assoc by simp
  finally show a·d·(b·d·(c·d))⁻¹ = a·(b·c)⁻¹·d⁻¹
    by simp
  from A2 have a·(b·c)·d = a·(b·(c·d))
    using group_op_closed group_oper_assoc by simp
  also from A1 A2 have ... =  a·(b·(d·c))
    using IsCommutative_def group_op_closed by simp
  also from A2 have ... =  a·b·d·c
    using group_op_closed group_oper_assoc by simp
  finally show a·(b·c)·d = a·b·d·c by simp
qed
```

Some other rearrangements with four elements.

```
lemma (in group0) group0_4_L8:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·(b·c)⁻¹ = (a·d⁻¹·c⁻¹)·(d·b⁻¹)
  a·b·(c·d) = c·a·(b·d)
  a·b·(c·d) = a·c·(b·d)
  a·(b·c⁻¹)·d = a·b·d·c⁻¹
  (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = a·c⁻¹
proof -
  from A2 have T:
    b·c ∈ G a·b ∈ G d⁻¹ ∈ G b⁻¹∈G c⁻¹∈G
    d⁻¹·b ∈ G c⁻¹·d ∈ G (b·c)⁻¹ ∈ G
    a·b ∈ G  (c·d)⁻¹ ∈ G  (b·d⁻¹)⁻¹ ∈ G  d·b⁻¹ ∈ G
    using group_op_closed inverse_in_group
    by auto
  from A2 have a·(b·c)⁻¹ = a·c⁻¹·b⁻¹ using group0_2_L14A by blast
  moreover from A2 have a·c⁻¹ = (a·d⁻¹)·(d·c⁻¹) using group0_2_L14A
    by blast
  ultimately have a·(b·c)⁻¹ = (a·d⁻¹)·(d·c⁻¹)·b⁻¹ by simp
  with A1 A2 T have a·(b·c)⁻¹= a·d⁻¹·(c⁻¹·d)·b⁻¹
    using IsCommutative_def by simp
  with A2 T show a·(b·c)⁻¹ = (a·d⁻¹·c⁻¹)·(d·b⁻¹)
    using group_op_closed group_oper_assoc by simp
  from A2 T have a·b·(c·d) = a·b·c·d
    using group_oper_assoc by simp
  also have a·b·c·d = c·a·b·d
  proof -
    from A1 A2 have a·b·c·d = c·(a·b)·d
      using IsCommutative_def group_op_closed
```

```
      by simp
    also from A2 have ... = c·a·b·d
      using group_op_closed group_oper_assoc
      by simp
    finally show thesis by simp
  qed
  also from A2 have c·a·b·d =  c·a·(b·d)
    using group_op_closed group_oper_assoc
    by simp
  finally show a·b·(c·d) = c·a·(b·d) by simp
  with A1 A2 show a·b·(c·d) = a·c·(b·d)
    using IsCommutative_def by simp
  from A1 A2 T show a·(b·c⁻¹)·d = a·b·d·c⁻¹
    using group0_4_L7 by simp
  from T have (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = (a·b)·((c·d)⁻¹·(b·d⁻¹)⁻¹)
    using group_oper_assoc by simp
  also from A1 A2 T have ... = (a·b)·(c⁻¹·d⁻¹·(d·b⁻¹))
    using group_inv_of_two group0_2_L12 IsCommutative_def
    by simp
  also from T have ... = (a·b)·(c⁻¹·(d⁻¹·(d·b⁻¹)))
    using group_oper_assoc by simp
  also from A1 A2 T have ... = a·c⁻¹
    using group_oper_assoc group0_2_L6 group0_2_L2 IsCommutative_def
    inv_cancel_two by simp
  finally show (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = a·c⁻¹
    by simp
qed
```

Some other rearrangements with four elements.

```
lemma (in group0) group0_4_L8A:
  assumes A1: P {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b⁻¹·(c·d⁻¹) = a·c·(b⁻¹·d⁻¹)
  a·b⁻¹·(c·d⁻¹) = a·c·b⁻¹·d⁻¹
proof -
  from A2 have
    T: a∈G  b⁻¹ ∈ G  c∈G  d⁻¹ ∈ G
    using inverse_in_group by auto
  with A1 show a·b⁻¹·(c·d⁻¹) = a·c·(b⁻¹·d⁻¹)
    by (rule group0_4_L8)
  with A2 T show  a·b⁻¹·(c·d⁻¹) = a·c·b⁻¹·d⁻¹
    using group_op_closed group_oper_assoc
    by simp
qed
```

Some rearrangements with an equation.

```
lemma (in group0) group0_4_L9:
  assumes A1: P {is commutative on} G
```

```
  and A2: a∈G  b∈G  c∈G  d∈G
  and A3: a = b·c⁻¹·d⁻¹
  shows
  d = b·a⁻¹·c⁻¹
  d = a⁻¹·b·c⁻¹
  b = a·d·c
proof -
  from A2 have T:
    a⁻¹ ∈ G  c⁻¹ ∈ G  d⁻¹ ∈ G  b·c⁻¹ ∈ G
    using group_op_closed inverse_in_group
    by auto
  with A2 A3 have a·(d⁻¹)⁻¹ =  b·c⁻¹
    using group0_2_L18 by simp
  with A2 have b·c⁻¹ = a·d
    using group_inv_of_inv by simp
  with A2 T have I: a⁻¹·(b·c⁻¹) = d
    using group0_2_L18 by simp
  with A1 A2 T show
    d = b·a⁻¹·c⁻¹
    d = a⁻¹·b·c⁻¹
    using group_oper_assoc IsCommutative_def by auto
  from A3 have a·d·c = (b·c⁻¹·d⁻¹)·d·c by simp
  also from A2 T have ... = b·c⁻¹·(d⁻¹·d)·c
    using group_oper_assoc by simp
  also from A2 T have ... =  b·c⁻¹·c
    using group0_2_L6 group0_2_L2 by simp
  also from A2 T have ... =  b·(c⁻¹·c)
    using group_oper_assoc by simp
  also from A2 have ... = b
    using group0_2_L6 group0_2_L2 by simp
  finally have a·d·c = b by simp
  thus b = a·d·c by simp
qed

end
```

37 Groups 2

```
theory Group_ZF_2 imports AbelianGroup_ZF func_ZF EquivClass1

begin
```

This theory continues `Group_ZF` and considers lifting the group structure to function spaces and projecting the group structure to quotient spaces, in particular the quotient qroup. We also define group homomorphisms and in particular the space `End(G,P)` of homomorphisms of a group into itself.

37.1 Lifting groups to function spaces

If we have a monoid (group) G than we get a monoid (group) structure on a space of functions valued in in G by defining $(f \cdot g)(x) := f(x) \cdot g(x)$. We call this process "lifting the monoid (group) to function space". This section formalizes this lifting.

The lifted operation is an operation on the function space.

```
lemma (in monoid0) Group_ZF_2_1_L0A:
  assumes A1: F = f {lifted to function space over} X
  shows F : (X→G)×(X→G)→(X→G)
proof -
  from monoidAssum have f : G×G→G
    using IsAmonoid_def IsAssociative_def by simp
  with A1 show thesis
    using func_ZF_1_L3 group0_1_L3B by auto
qed
```

The result of the lifted operation is in the function space.

```
lemma (in monoid0) Group_ZF_2_1_L0:
  assumes A1:F = f {lifted to function space over} X
  and A2:s:X→G r:X→G
  shows F⟨ s,r⟩ : X→G
proof -
  from A1 have F : (X→G)×(X→G)→(X→G)
    using Group_ZF_2_1_L0A
    by simp
  with A2 show thesis using apply_funtype
    by simp
qed
```

The lifted monoid operation has a neutral element, namely the constant function with the neutral element as the value.

```
lemma (in monoid0) Group_ZF_2_1_L1:
  assumes A1: F = f {lifted to function space over} X
  and A2: E = ConstantFunction(X,TheNeutralElement(G,f))
  shows E : X→G ∧ (∀s∈X→G. F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s)
proof
  from A2 show T1:E : X→G
    using unit_is_neutral func1_3_L1 by simp
  show ∀s∈X→G. F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s
  proof
    fix s assume A3:s:X→G
    from monoidAssum have T2:f : G×G→G
      using IsAmonoid_def IsAssociative_def by simp
    from A3 A1 T1 have
      F⟨ E,s⟩ : X→G F⟨ s,E⟩ : X→G s : X→G
      using Group_ZF_2_1_L0 by auto
```

```
    moreover from T2 A1 T1 A2 A3 have
      ∀x∈X. (F⟨ E,s⟩)(x) = s(x)
      ∀x∈X. (F⟨ s,E⟩)(x) = s(x)
      using func_ZF_1_L4 group0_1_L3B func1_3_L2
 apply_type unit_is_neutral by auto
    ultimately show
      F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s
      using fun_extension_iff by auto
  qed
qed
```

Monoids can be lifted to a function space.

```
lemma (in monoid0) Group_ZF_2_1_T1:
  assumes A1: F = f {lifted to function space over} X
  shows IsAmonoid(X→G,F)
proof -
  from monoidAssum A1 have
    F {is associative on} (X→G)
    using IsAmonoid_def func_ZF_2_L4 group0_1_L3B
    by auto
  moreover from A1 have
    ∃ E ∈ X→G. ∀s ∈ X→G. F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s
    using Group_ZF_2_1_L1 by blast
  ultimately show thesis using IsAmonoid_def
    by simp
qed
```

The constant function with the neutral element as the value is the neutral element of the lifted monoid.

```
lemma Group_ZF_2_1_L2:
  assumes A1: IsAmonoid(G,f)
  and A2: F = f {lifted to function space over} X
  and A3: E = ConstantFunction(X,TheNeutralElement(G,f))
  shows E = TheNeutralElement(X→G,F)
proof -
  from A1 A2 have
     T1:monoid0(G,f) and T2:monoid0(X→G,F)
    using monoid0_def monoid0.Group_ZF_2_1_T1
    by auto
  from T1 A2 A3 have
    E : X→G ∧ (∀s∈X→G. F⟨ E,s⟩ = s ∧ F⟨ s,E⟩ = s)
    using monoid0.Group_ZF_2_1_L1 by simp
  with T2 show thesis
    using monoid0.group0_1_L4 by auto
qed
```

The lifted operation acts on the functions in a natural way defined by the monoid operation.

```
lemma (in monoid0) lifted_val:
```

```
  assumes F = f {lifted to function space over} X
  and s:X→G  r:X→G
  and x∈X
  shows (F⟨s,r⟩)(x) = s(x) ⊕ r(x)
  using monoidAssum assms IsAmonoid_def IsAssociative_def
      group0_1_L3B func_ZF_1_L4
  by auto
```

The lifted operation acts on the functions in a natural way defined by the group operation. This is the same as `lifted_val`, but in the `group0` context.

```
lemma (in group0) Group_ZF_2_1_L3:
  assumes F = P {lifted to function space over} X
  and s:X→G r:X→G
  and x∈X
  shows (F⟨s,r⟩)(x) = s(x)·r(x)
  using assms group0_2_L1 monoid0.lifted_val by simp
```

In the group0 context we can apply theorems proven in monoid0 context to the lifted monoid.

```
lemma (in group0) Group_ZF_2_1_L4:
  assumes A1: F = P {lifted to function space over} X
  shows monoid0(X→G,F)
proof -
  from A1 show thesis
    using group0_2_L1 monoid0.Group_ZF_2_1_T1 monoid0_def
    by simp
qed
```

The compostion of a function $f : X \to G$ with the group inverse is a right inverse for the lifted group.

```
lemma (in group0) Group_ZF_2_1_L5:
  assumes A1: F = P {lifted to function space over} X
  and A2: s : X→G
  and A3: i = GroupInv(G,P) O s
  shows i: X→G and F⟨ s,i⟩ = TheNeutralElement(X→G,F)
proof -
  let E = ConstantFunction(X,1)
  have E : X→G
    using group0_2_L2 func1_3_L1 by simp
  moreover from groupAssum A2 A3 A1 have
    F⟨ s,i⟩ :  X→G using group0_2_T2 comp_fun
      Group_ZF_2_1_L4 monoid0.group0_1_L1
    by simp
  moreover from groupAssum A2 A3 A1 have
    ∀x∈X. (F⟨ s,i⟩)(x) = E(x)
    using group0_2_T2 comp_fun Group_ZF_2_1_L3
      comp_fun_apply apply_funtype group0_2_L6 func1_3_L2
    by simp
```

```
  moreover from groupAssum A1 have
    E = TheNeutralElement(X→G,F)
    using IsAgroup_def Group_ZF_2_1_L2 by simp
  ultimately show F⟨ s,i⟩ = TheNeutralElement(X→G,F)
    using fun_extension_iff IsAgroup_def Group_ZF_2_1_L2
    by simp
  from groupAssum A2 A3 show i: X→G
    using group0_2_T2 comp_fun by simp
qed
```

Groups can be lifted to the function space.

```
theorem (in group0) Group_ZF_2_1_T2:
  assumes A1: F = P {lifted to function space over} X
  shows IsAgroup(X→G,F)
proof -
  from A1 have IsAmonoid(X→G,F)
    using group0_2_L1 monoid0.Group_ZF_2_1_T1
    by simp
  moreover have
    ∀s∈X→G. ∃i∈X→G. F⟨ s,i⟩ = TheNeutralElement(X→G,F)
  proof
    fix s assume A2: s : X→G
    let i = GroupInv(G,P) O s
    from groupAssum A2 have i:X→G
      using group0_2_T2 comp_fun by simp
    moreover from A1 A2 have
      F⟨ s,i⟩ = TheNeutralElement(X→G,F)
      using Group_ZF_2_1_L5 by fast
   ultimately show ∃i∈X→G. F⟨ s,i⟩ = TheNeutralElement(X→G,F)
      by auto
  qed
  ultimately show thesis using IsAgroup_def
    by simp
qed
```

The propositions proven in the `group0` context are valid in the same context when applied to the function space with the lifted group operation.

```
lemma (in group0) group0_valid_fun_space:
  shows group0(X→G,P {lifted to function space over} X)
  using Group_ZF_2_1_T2 unfolding group0_def by simp
```

What is the group inverse for the lifted group?

```
lemma (in group0) Group_ZF_2_1_L6:
  assumes A1: F = P {lifted to function space over} X
  shows ∀s∈(X→G). GroupInv(X→G,F)(s) = GroupInv(G,P) O s
proof -
  from A1 have  group0(X→G,F)
    using group0_def Group_ZF_2_1_T2
    by simp
```

```
  moreover from A1 have ∀s∈X→G. GroupInv(G,P) O s : X→G ∧
    F⟨ s,GroupInv(G,P) O s⟩ = TheNeutralElement(X→G,F)
    using Group_ZF_2_1_L5 by simp
  ultimately have
    ∀s∈(X→G).  GroupInv(G,P) O s = GroupInv(X→G,F)(s)
    by (rule group0.group0_2_L9A)
  thus thesis by simp
qed
```

What is the value of the group inverse for the lifted group?

```
corollary (in group0) lift_gr_inv_val:
  assumes F = P {lifted to function space over} X and
  s : X→G and x∈X
  shows  (GroupInv(X→G,F)(s))(x) = (s(x))⁻¹
  using groupAssum assms Group_ZF_2_1_L6 group0_2_T2 comp_fun_apply
  by simp
```

What is the group inverse in a subgroup of the lifted group?

```
lemma (in group0) Group_ZF_2_1_L6A:
  assumes A1: F = P {lifted to function space over} X
  and A2: IsAsubgroup(H,F)
  and A3: g = restrict(F,H×H)
  and A4: s∈H
  shows GroupInv(H,g)(s) = GroupInv(G,P) O s
proof -
  from A1 have T1: group0(X→G,F)
    using group0_def Group_ZF_2_1_T2
    by simp
  with A2 A3 A4 have GroupInv(H,g)(s) = GroupInv(X→G,F)(s)
    using group0.group0_3_T1 restrict by simp
  moreover from T1 A1 A2 A4 have
    GroupInv(X→G,F)(s) = GroupInv(G,P) O s
    using group0.group0_3_L2 Group_ZF_2_1_L6 by blast
  ultimately show thesis by simp
qed
```

The neutral element of a subgroup of the lifted group is the constant function with value equal to the neutral element of the group.

```
lemma (in group0) lift_group_subgr_neut:
  assumes F = P {lifted to function space over} X and IsAsubgroup(H,F)
  shows TheNeutralElement(H,restrict(F,H×H)) = ConstantFunction(X,1)
proof -
  from assms have
    TheNeutralElement(H,restrict(F,H×H)) = TheNeutralElement(X→G,F)
    using group0_valid_fun_space group0.group0_3_L4 by blast
  also from groupAssum assms(1) have ... = ConstantFunction(X,1)
    using Group_ZF_2_1_L2 unfolding IsAgroup_def by simp
  finally show thesis by simp
qed
```

If a group is abelian, then its lift to a function space is also abelian.

```
lemma (in group0) Group_ZF_2_1_L7:
  assumes A1: F = P {lifted to function space over} X
  and A2: P {is commutative on} G
  shows F {is commutative on} (X→G)
proof-
  from A1 A2  have
    F {is commutative on} (X→range(P))
    using group_oper_fun func_ZF_2_L2
    by simp
  moreover from groupAssum have range(P) = G
    using group0_2_L1 monoid0.group0_1_L3B
    by simp
  ultimately show thesis by simp
qed
```

37.2 Equivalence relations on groups

The goal of this section is to establish that (under some conditions) given an equivalence relation on a group or (monoid)we can project the group (monoid) structure on the quotient and obtain another group.

The neutral element class is neutral in the projection.

```
lemma (in monoid0) Group_ZF_2_2_L1:
  assumes A1: equiv(G,r) and A2:Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  and A4: e = TheNeutralElement(G,f)
  shows r{e} ∈ G//r ∧
  (∀c ∈ G//r. F⟨ r{e},c⟩ = c ∧  F⟨ c,r{e}⟩ = c)
proof
  from A4 show T1:r{e} ∈ G//r
    using unit_is_neutral quotientI
    by simp
  show
    ∀c ∈ G//r. F⟨ r{e},c⟩ = c ∧  F⟨ c,r{e}⟩ = c
  proof
    fix c assume A5:c ∈ G//r
    then obtain g where D1:g∈G c = r{g}
      using quotient_def by auto
    with A1 A2 A3 A4 D1 show
      F⟨ r{e},c⟩ = c ∧  F⟨ c,r{e}⟩ = c
      using unit_is_neutral EquivClass_1_L10
      by simp
  qed
qed
```

The projected structure is a monoid.

```
theorem (in monoid0) Group_ZF_2_2_T1:
```

```
  assumes A1: equiv(G,r) and A2: Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  shows IsAmonoid(G//r,F)
proof -
  let E = r{TheNeutralElement(G,f)}
  from A1 A2 A3 have
    E ∈ G//r ∧ (∀c∈G//r. F⟨ E,c⟩ = c ∧ F⟨ c,E⟩ = c)
    using Group_ZF_2_2_L1 by simp
  hence
    ∃E∈G//r. ∀ c∈G//r. F⟨ E,c⟩ = c ∧ F⟨ c,E⟩ = c
    by auto
  with monoidAssum A1 A2 A3 show thesis
    using IsAmonoid_def EquivClass_2_T2
    by simp
qed
```

The class of the neutral element is the neutral element of the projected monoid.

```
lemma Group_ZF_2_2_L1:
  assumes A1: IsAmonoid(G,f)
  and A2: equiv(G,r) and A3: Congruent2(r,f)
  and A4: F = ProjFun2(G,r,f)
  and A5: e = TheNeutralElement(G,f)
  shows  r{e} = TheNeutralElement(G//r,F)
proof -
  from A1 A2 A3 A4 have
    T1:monoid0(G,f) and T2:monoid0(G//r,F)
    using monoid0_def monoid0.Group_ZF_2_2_T1 by auto
  from T1 A2 A3 A4 A5 have r{e} ∈ G//r ∧
    (∀c ∈ G//r. F⟨ r{e},c⟩ = c ∧  F⟨ c,r{e}⟩ = c)
    using monoid0.Group_ZF_2_2_L1 by simp
  with T2 show thesis using monoid0.group0_1_L4
    by auto
qed
```

The projected operation can be defined in terms of the group operation on representants in a natural way.

```
lemma (in group0) Group_ZF_2_2_L2:
  assumes A1: equiv(G,r) and A2: Congruent2(r,P)
  and A3: F = ProjFun2(G,r,P)
  and A4: a∈G b∈G
  shows F⟨ r{a},r{b}⟩ = r{a·b}
proof -
  from A1 A2 A3 A4 show thesis
    using EquivClass_1_L10 by simp
qed
```

The class of the inverse is a right inverse of the class.

```
lemma (in group0) Group_ZF_2_2_L3:
```

```
  assumes A1: equiv(G,r) and A2: Congruent2(r,P)
  and A3: F = ProjFun2(G,r,P)
  and A4: a∈G
  shows F⟨r{a},r{a⁻¹}⟩ = TheNeutralElement(G//r,F)
proof -
  from A1 A2 A3 A4 have
    F⟨r{a},r{a⁻¹}⟩ = r{1}
    using inverse_in_group Group_ZF_2_2_L2 group0_2_L6
    by simp
  with groupAssum A1 A2 A3 show thesis
    using IsAgroup_def Group_ZF_2_2_L1 by simp
qed
```

The group structure can be projected to the quotient space.

```
theorem (in group0) Group_ZF_3_T2:
  assumes A1: equiv(G,r) and A2: Congruent2(r,P)
  shows IsAgroup(G//r,ProjFun2(G,r,P))
proof -
  let F = ProjFun2(G,r,P)
  let E = TheNeutralElement(G//r,F)
  from groupAssum A1 A2 have IsAmonoid(G//r,F)
    using IsAgroup_def monoid0_def monoid0.Group_ZF_2_2_T1
    by simp
  moreover have
    ∀c∈G//r. ∃b∈G//r. F⟨ c,b⟩ = E
  proof
    fix c assume A3: c ∈ G//r
    then obtain g where D1: g∈G  c = r{g}
      using quotient_def by auto
    let b = r{g⁻¹}
    from D1 have b ∈ G//r
      using inverse_in_group quotientI
      by simp
    moreover from A1 A2 D1 have
      F⟨ c,b⟩ = E
      using Group_ZF_2_2_L3 by simp
    ultimately show ∃b∈G//r. F⟨ c,b⟩ = E
      by auto
  qed
  ultimately show thesis
    using IsAgroup_def by simp
qed
```

The group inverse (in the projected group) of a class is the class of the inverse.

```
lemma (in group0) Group_ZF_2_2_L4:
  assumes A1: equiv(G,r) and
  A2: Congruent2(r,P) and
  A3: F = ProjFun2(G,r,P) and
```

```
  A4: a∈G
  shows r{a⁻¹} = GroupInv(G//r,F)(r{a})
proof -
  from A1 A2 A3 have group0(G//r,F)
    using Group_ZF_3_T2 group0_def by simp
  moreover from A4 have
    r{a} ∈ G//r  r{a⁻¹} ∈ G//r
    using inverse_in_group quotientI by auto
  moreover from A1 A2 A3 A4 have
     F⟨r{a},r{a⁻¹}⟩ = TheNeutralElement(G//r,F)
    using Group_ZF_2_2_L3 by simp
  ultimately show thesis
    by (rule group0.group0_2_L9)
qed
```

37.3 Normal subgroups and quotient groups

If H is a subgroup of G, then for every $a \in G$ we can cosider the sets $\{a \cdot h.h \in H\}$ and $\{h \cdot a.h \in H\}$ (called a left and right "coset of H", resp.) These sets sometimes form a group, called the "quotient group". This section discusses the notion of quotient groups.

A normal subgorup N of a group G is such that aba^{-1} belongs to N if $a \in G, b \in N$.

```
definition
  IsAnormalSubgroup(G,P,N) ≡ IsAsubgroup(N,P) ∧
  (∀n∈N.∀g∈G. P⟨  P⟨  g,n ⟩,GroupInv(G,P)(g) ⟩ ∈ N)
```

Having a group and a normal subgroup N we can create another group consisting of eqivalence classes of the relation $a \sim b \equiv a \cdot b^{-1} \in N$. We will refer to this relation as the quotient group relation. The classes of this relation are in fact cosets of subgroup H.

```
definition
  QuotientGroupRel(G,P,H) ≡
  {⟨ a,b⟩ ∈ G×G. P⟨ a, GroupInv(G,P)(b)⟩ ∈ H}
```

Next we define the operation in the quotient group as the projection of the group operation on the classses of the quotient group relation.

```
definition
  QuotientGroupOp(G,P,H) ≡ ProjFun2(G,QuotientGroupRel(G,P,H ),P)
```

Definition of a normal subgroup in a more readable notation.

```
lemma (in group0) Group_ZF_2_4_L0:
  assumes IsAnormalSubgroup(G,P,H)
  and g∈G n∈H
  shows g·n·g⁻¹ ∈ H
  using assms IsAnormalSubgroup_def by simp
```

The quotient group relation is reflexive.

```
lemma (in group0) Group_ZF_2_4_L1:
  assumes IsAsubgroup(H,P)
  shows refl(G,QuotientGroupRel(G,P,H))
  using assms  group0_2_L6 group0_3_L5
    QuotientGroupRel_def refl_def by simp
```

The quotient group relation is symmetric.

```
lemma (in group0) Group_ZF_2_4_L2:
  assumes A1:IsAsubgroup(H,P)
  shows sym(QuotientGroupRel(G,P,H))
proof -
  {
    fix a b assume A2: ⟨ a,b⟩ ∈ QuotientGroupRel(G,P,H)
    with A1 have (a·b⁻¹)⁻¹ ∈ H
      using QuotientGroupRel_def group0_3_T3A
      by simp
    moreover from A2 have (a·b⁻¹)⁻¹ =  b·a⁻¹
      using QuotientGroupRel_def group0_2_L12
      by simp
    ultimately have b·a⁻¹ ∈ H by simp
    with A2 have ⟨ b,a⟩ ∈ QuotientGroupRel(G,P,H)
      using QuotientGroupRel_def by simp
  }
  then show thesis using symI by simp
qed
```

The quotient group relation is transistive.

```
lemma (in group0) Group_ZF_2_4_L3A:
  assumes A1: IsAsubgroup(H,P) and
  A2: ⟨ a,b⟩ ∈ QuotientGroupRel(G,P,H) and
  A3: ⟨ b,c⟩ ∈ QuotientGroupRel(G,P,H)
  shows ⟨ a,c⟩ ∈ QuotientGroupRel(G,P,H)
proof -
  let r = QuotientGroupRel(G,P,H)
  from A2 A3 have T1:a∈G b∈G c∈G
    using QuotientGroupRel_def by auto
  from A1 A2 A3 have (a·b⁻¹)·(b·c⁻¹) ∈ H
    using  QuotientGroupRel_def group0_3_L6
    by simp
  moreover from T1 have
    a·c⁻¹ = (a·b⁻¹)·(b·c⁻¹)
    using group0_2_L14A by blast
  ultimately have a·c⁻¹ ∈ H
    by simp
  with T1 show thesis using QuotientGroupRel_def
    by simp
qed
```

The quotient group relation is an equivalence relation. Note we do not need the subgroup to be normal for this to be true.

```
lemma (in group0) Group_ZF_2_4_L3: assumes A1:IsAsubgroup(H,P)
  shows equiv(G,QuotientGroupRel(G,P,H))
proof -
  let r = QuotientGroupRel(G,P,H)
  from A1 have
     ∀a b c. (⟨a, b⟩ ∈ r  ∧  ⟨b, c⟩ ∈ r ⟶ ⟨a, c⟩ ∈ r)
    using Group_ZF_2_4_L3A by blast
  then have trans(r)
    using Fol1_L2 by blast
  with A1 show thesis
    using Group_ZF_2_4_L1 Group_ZF_2_4_L2
      QuotientGroupRel_def equiv_def
    by auto
qed
```

The next lemma states the essential condition for congruency of the group operation with respect to the quotient group relation.

```
lemma (in group0) Group_ZF_2_4_L4:
  assumes A1: IsAnormalSubgroup(G,P,H)
  and A2: ⟨a1,a2⟩ ∈ QuotientGroupRel(G,P,H)
  and A3: ⟨b1,b2⟩ ∈ QuotientGroupRel(G,P,H)
  shows ⟨a1·b1, a2·b2⟩ ∈ QuotientGroupRel(G,P,H)
proof -
  from A2 A3 have T1:
    a1∈G  a2∈G  b1∈G  b2∈G
    a1·b1 ∈ G  a2·b2 ∈ G
    b1·b2⁻¹ ∈ H  a1·a2⁻¹ ∈ H
    using QuotientGroupRel_def group0_2_L1 monoid0.group0_1_L1
    by auto
  with A1 show thesis using
    IsAnormalSubgroup_def group0_3_L6 group0_2_L15
    QuotientGroupRel_def by simp
qed
```

If the subgroup is normal, the group operation is congruent with respect to the quotient group relation.

```
lemma Group_ZF_2_4_L5A:
  assumes IsAgroup(G,P)
  and IsAnormalSubgroup(G,P,H)
  shows Congruent2(QuotientGroupRel(G,P,H),P)
  using assms group0_def group0.Group_ZF_2_4_L4 Congruent2_def
  by simp
```

The quotient group is indeed a group.

```
theorem Group_ZF_2_4_T1:
  assumes IsAgroup(G,P) and IsAnormalSubgroup(G,P,H)
```

```
  shows
  IsAgroup(G//QuotientGroupRel(G,P,H),QuotientGroupOp(G,P,H))
  using assms group0_def group0.Group_ZF_2_4_L3 IsAnormalSubgroup_def
    Group_ZF_2_4_L5A group0.Group_ZF_3_T2 QuotientGroupOp_def
  by simp
```

The class (coset) of the neutral element is the neutral element of the quotient group.

```
lemma Group_ZF_2_4_L5B:
  assumes IsAgroup(G,P) and IsAnormalSubgroup(G,P,H)
  and r = QuotientGroupRel(G,P,H)
  and e = TheNeutralElement(G,P)
  shows  r{e} = TheNeutralElement(G//r,QuotientGroupOp(G,P,H))
  using assms IsAnormalSubgroup_def group0_def
    IsAgroup_def group0.Group_ZF_2_4_L3 Group_ZF_2_4_L5A
    QuotientGroupOp_def Group_ZF_2_2_L1
  by simp
```

A group element is equivalent to the neutral element iff it is in the subgroup we divide the group by.

```
lemma (in group0) Group_ZF_2_4_L5C: assumes a∈G
  shows ⟨a,1⟩ ∈ QuotientGroupRel(G,P,H) ⟷ a∈H
  using assms QuotientGroupRel_def group_inv_of_one group0_2_L2
  by auto
```

A group element is in H iff its class is the neutral element of G/H.

```
lemma (in group0) Group_ZF_2_4_L5D:
  assumes A1: IsAnormalSubgroup(G,P,H) and
  A2: a∈G and
  A3: r = QuotientGroupRel(G,P,H) and
  A4: TheNeutralElement(G//r,QuotientGroupOp(G,P,H)) = e
  shows r{a} = e ⟷ ⟨a,1⟩ ∈ r
proof
  assume r{a} = e
  with groupAssum assms have
    r{1} = r{a} and I: equiv(G,r)
    using Group_ZF_2_4_L5B IsAnormalSubgroup_def Group_ZF_2_4_L3
    by auto
  with A2 have ⟨1,a⟩ ∈ r using eq_equiv_class
    by simp
  with I show ⟨a,1⟩ ∈ r by (rule equiv_is_sym)
next assume ⟨a,1⟩ ∈ r
  moreover from A1 A3 have equiv(G,r)
    using IsAnormalSubgroup_def Group_ZF_2_4_L3
    by simp
  ultimately have r{a} = r{1}
    using equiv_class_eq by simp
  with groupAssum A1 A3 A4 show r{a} = e
```

```
    using Group_ZF_2_4_L5B by simp
qed
```

The class of $a \in G$ is the neutral element of the quotient G/H iff $a \in H$.

```
lemma (in group0) Group_ZF_2_4_L5E:
  assumes IsAnormalSubgroup(G,P,H) and
  a∈G and r = QuotientGroupRel(G,P,H) and
  TheNeutralElement(G//r,QuotientGroupOp(G,P,H)) = e
  shows r{a} = e ⟷ a∈H
  using assms Group_ZF_2_4_L5C  Group_ZF_2_4_L5D
  by simp
```

Essential condition to show that every subgroup of an abelian group is normal.

```
lemma (in group0) Group_ZF_2_4_L5:
  assumes A1: P {is commutative on} G
  and A2: IsAsubgroup(H,P)
  and A3: g∈G  h∈H
  shows g·h·g⁻¹ ∈ H
proof -
  from A2 A3 have T1:h∈G g⁻¹ ∈ G
    using group0_3_L2 inverse_in_group by auto
  with A3 A1 have g·h·g⁻¹ = g⁻¹·g·h
    using group0_4_L4A by simp
  with A3 T1 show thesis using
    group0_2_L6 group0_2_L2
    by simp
qed
```

Every subgroup of an abelian group is normal. Moreover, the quotient group is also abelian.

```
lemma Group_ZF_2_4_L6:
  assumes A1: IsAgroup(G,P)
  and A2: P {is commutative on} G
  and A3: IsAsubgroup(H,P)
  shows  IsAnormalSubgroup(G,P,H)
  QuotientGroupOp(G,P,H) {is commutative on} (G//QuotientGroupRel(G,P,H))
proof -
  from A1 A2 A3 show T1: IsAnormalSubgroup(G,P,H) using
    group0_def IsAnormalSubgroup_def group0.Group_ZF_2_4_L5
    by simp
  let r = QuotientGroupRel(G,P,H)
  from A1 A3 T1 have equiv(G,r) Congruent2(r,P)
    using group0_def group0.Group_ZF_2_4_L3 Group_ZF_2_4_L5A
    by auto
  with A2 show
    QuotientGroupOp(G,P,H) {is commutative on} (G//QuotientGroupRel(G,P,H))
    using EquivClass_2_T1 QuotientGroupOp_def
```

```
  by simp
qed
```

The group inverse (in the quotient group) of a class (coset) is the class of the inverse.

```
lemma (in group0) Group_ZF_2_4_L7:
 assumes IsAnormalSubgroup(G,P,H)
 and a∈G and r = QuotientGroupRel(G,P,H)
 and F = QuotientGroupOp(G,P,H)
 shows r{a⁻¹} = GroupInv(G//r,F)(r{a})
 using groupAssum assms IsAnormalSubgroup_def Group_ZF_2_4_L3
   Group_ZF_2_4_L5A QuotientGroupOp_def Group_ZF_2_2_L4
 by simp
```

37.4 Function spaces as monoids

On every space of functions $\{f : X \to X\}$ we can define a natural monoid structure with composition as the operation. This section explores this fact.

The next lemma states that composition has a neutral element, namely the identity function on X (the one that maps $x \in X$ into itself).

```
lemma Group_ZF_2_5_L1: assumes A1: F = Composition(X)
 shows ∃I∈(X→X). ∀f∈(X→X). F⟨ I,f⟩ = f ∧ F⟨ f,I⟩ = f
proof-
 let I = id(X)
 from A1 have
   I ∈ X→X ∧ (∀f∈(X→X). F⟨ I,f⟩ = f ∧ F⟨ f,I⟩ = f)
   using id_type func_ZF_6_L1A by simp
 thus thesis by auto
qed
```

The space of functions that map a set X into itsef is a monoid with composition as operation and the identity function as the neutral element.

```
lemma Group_ZF_2_5_L2: shows
 IsAmonoid(X→X,Composition(X))
 id(X) = TheNeutralElement(X→X,Composition(X))
proof -
 let I = id(X)
 let F = Composition(X)
 show IsAmonoid(X→X,Composition(X))
   using func_ZF_5_L5 Group_ZF_2_5_L1 IsAmonoid_def
   by auto
 then have monoid0(X→X,F)
   using monoid0_def by simp
 moreover have
   I ∈ X→X ∧ (∀f∈(X→X). F⟨ I,f⟩ = f ∧ F⟨ f,I⟩ = f)
   using id_type func_ZF_6_L1A by simp
 ultimately show I = TheNeutralElement(X→X,F)
```

```
    using monoid0.group0_1_L4 by auto
qed
```

37.5 Homomorphisms

A homomorphism is a function between groups that preserves the group operations.

In general we may have a homomorphism not only between groups, but also between various algebraic structures with one operation like magmas, semigroups, quasigroups, loops and monoids. In all cases the homomorphism is defined by using the morphism property. In the multiplicative notation we we will write that f has a morphism property if $f(x \cdot_G y) = f(x) \cdot_H f(y)$ for all $x, y \in G$. Below we write this definition in raw set theory notation and use the expression `IsMorphism` instead of the possible, but longer `HasMorphismProperty`.

```
definition
  IsMorphism(G,P,F,f) ≡ ∀g₁∈G. ∀g₂∈G. f(P⟨g₁,g₂⟩) = F⟨f(g₁),f(g₂)⟩
```

A function $f : G \to H$ between algebraic structures $(G, \cdot_G)$ and $(H, \cdot_H)$ with one operation (each) is a homomorphism if it has the morphism property.

```
definition
  Homomor(f,G,P,H,F) ≡ f:G→H ∧ IsMorphism(G,P,F,f)
```

Now a lemma about the definition:

```
lemma homomor_eq:
  assumes Homomor(f,G,P,H,F) g₁∈G g₂∈G
  shows f(P⟨g₁,g₂⟩) = F⟨f(g₁),f(g₂)⟩
  using assms unfolding Homomor_def IsMorphism_def by auto
```

An endomorphism is a homomorphism from a group to the same group. We define `End(G,P)` as the set of endomorphisms for a given group. As we show later when the group is abelian, the set of endomorphisms with pointwise adddition and composition as multiplication forms a ring.

```
definition
  End(G,P) ≡ {f∈G→G. Homomor(f,G,P,G,P)}
```

The defining property of an endomorphism written in notation used in `group0` context:

```
lemma (in group0) endomor_eq: assumes f ∈ End(G,P) g₁∈G g₂∈G
  shows f(g₁·g₂) = f(g₁)·f(g₂)
  using assms homomor_eq unfolding End_def by auto
```

A function that maps a group G into itself and satisfies $f(g_1 \cdot g2) = f(g_1) \cdot f(g_2)$ is an endomorphism.

```
lemma (in group0) eq_endomor:
```

```
  assumes f:G→G and ∀g₁∈G. ∀g₂∈G. f(g₁·g₂)=f(g₁)·f(g₂)
  shows f ∈ End(G,P)
  using assms  unfolding End_def Homomor_def IsMorphism_def by simp
```

The set of endomorphisms forms a submonoid of the monoid of function from a set to that set under composition.

```
lemma (in group0) end_composition:
  assumes f₁∈End(G,P) f₂∈End(G,P)
  shows Composition(G)⟨f₁,f₂⟩ ∈ End(G,P)
proof-
  from assms have fun: f₁:G→G f₂:G→G unfolding End_def by auto
  then have f₁ O f₂:G→G using comp_fun by auto
  from assms fun(2) have
    ∀g₁∈G. ∀g₂∈G. (f₁ O f₂)(g₁·g₂) = ((f₁ O f₂)(g₁))·((f₁ O f₂)(g₂))
    using group_op_closed comp_fun_apply endomor_eq apply_type
    by simp
  with fun ‹f₁ O f₂:G→G› show thesis using eq_endomor func_ZF_5_L2
    by simp
qed
```

We will use some binary operations that are naturally defined on the function space $G \to G$, but we consider them restricted to the endomorphisms of G. To shorten the notation in such case we define an abbreviation `InEnd(F,G,P)` which restricts a binary operation F to the set of endomorphisms of G.

```
abbreviation InEnd(_ {in End} [_,_])
  where InEnd(F,G,P) ≡ restrict(F,End(G,P)×End(G,P))
```

Endomoprhisms of a group form a monoid with composition as the binary operation, and the identity map as the neutral element.

```
theorem (in group0) end_comp_monoid:
  shows IsAmonoid(End(G,P),InEnd(Composition(G),G,P))
  and TheNeutralElement(End(G,P),InEnd(Composition(G),G,P)) = id(G)
proof -
  let C₀ = InEnd(Composition(G),G,P)
  have fun: id(G):G→G unfolding id_def by auto
  { fix g h assume g∈Gh∈G
    then have id(G)(g·h)=(id(G)g)·(id(G)h)
      using group_op_closed by simp
  }
  with groupAssum fun have id(G) ∈ End(G,P) using eq_endomor by simp

  moreover  have A0: id(G)=TheNeutralElement(G → G, Composition(G))

    using Group_ZF_2_5_L2(2) by auto
  ultimately have A1: TheNeutralElement(G → G, Composition(G)) ∈ End(G,P)
by auto
  moreover have A2: End(G,P) ⊆ G→G unfolding End_def by blast
  moreover have A3: End(G,P) {is closed under} Composition(G)
```

```
    using end_composition unfolding IsOpClosed_def by blast
  ultimately show IsAmonoid(End(G,P),C0)
    using monoid0.group0_1_T1 Group_ZF_2_5_L2(1) unfolding monoid0_def
    by blast
  have IsAmonoid(G→G,Composition(G)) using Group_ZF_2_5_L2(1) by auto
  with A0 A1 A2 A3 show TheNeutralElement(End(G,P),C0) = id(G)
    using group0_1_L6 by auto
qed

end
```

38 Groups 3

```
theory Group_ZF_3 imports Group_ZF_2 Finite1

begin
```

In this theory we consider notions in group theory that are useful for the construction of real numbers in the Real_ZF_x series of theories.

38.1 Group valued finite range functions

In this section show that the group valued functions $f : X \to G$, with the property that $f(X)$ is a finite subset of G, is a group. Such functions play an important role in the construction of real numbers in the Real_ZF series.

The following proves the essential condition to show that the set of finite range functions is closed with respect to the lifted group operation.

```
lemma (in group0) Group_ZF_3_1_L1:
  assumes A1: F = P {lifted to function space over} X
  and
  A2: s ∈ FinRangeFunctions(X,G)  r ∈  FinRangeFunctions(X,G)
  shows F⟨ s,r⟩ ∈ FinRangeFunctions(X,G)
proof -
  let q = F⟨ s,r⟩
  from A2 have T1:s:X→G r:X→G
    using FinRangeFunctions_def by auto
  with A1 have T2:q : X→G
    using group0_2_L1 monoid0.Group_ZF_2_1_L0
    by simp
  moreover have q(X) ∈ Fin(G)
  proof -
    from A2 have
      {s(x). x∈X} ∈ Fin(G)
      {r(x). x∈X} ∈ Fin(G)
      using Finite1_L18 by auto
    with A1 T1 T2 show thesis using
      group_oper_fun Finite1_L15 Group_ZF_2_1_L3 func_imagedef
```

```
      by simp
  qed
  ultimately show thesis using FinRangeFunctions_def
    by simp
qed
```

The set of group valued finite range functions is closed with respect to the lifted group operation.

```
lemma (in group0) Group_ZF_3_1_L2:
  assumes A1: F = P {lifted to function space over} X
  shows FinRangeFunctions(X,G) {is closed under} F
proof -
  let A = FinRangeFunctions(X,G)
  from A1 have ∀x∈A. ∀y∈A. F⟨ x,y⟩ ∈ A
    using Group_ZF_3_1_L1 by simp
  then show thesis using IsOpClosed_def by simp
qed
```

A composition of a finite range function with the group inverse is a finite range function.

```
lemma (in group0) Group_ZF_3_1_L3:
  assumes A1: s ∈ FinRangeFunctions(X,G)
  shows GroupInv(G,P) O s ∈ FinRangeFunctions(X,G)
  using groupAssum assms group0_2_T2 Finite1_L20 by simp
```

The set of finite range functions is a subgroup of the lifted group.

```
theorem Group_ZF_3_1_T1:
  assumes A1: IsAgroup(G,P)
  and A2: F = P {lifted to function space over} X
  and A3: X≠0
  shows IsAsubgroup(FinRangeFunctions(X,G),F)
proof -
  let e = TheNeutralElement(G,P)
  let S = FinRangeFunctions(X,G)
  from A1 have T1: group0(G,P) using group0_def
    by simp
  with A1 A2 have T2:group0(X→G,F)
    using group0.Group_ZF_2_1_T2 group0_def
    by simp
  moreover have S ≠ 0
  proof -
    from T1 A3 have
      ConstantFunction(X,e) ∈ S
      using group0.group0_2_L1 monoid0.unit_is_neutral
 Finite1_L17 by simp
    thus thesis by auto
  qed
  moreover have S ⊆ X→G
```

```
    using FinRangeFunctions_def by auto
  moreover from A2 T1 have
    S {is closed under} F
    using group0.Group_ZF_3_1_L2
    by simp
  moreover from A1 A2 T1 have
    ∀s ∈ S. GroupInv(X→G,F)(s) ∈ S
    using FinRangeFunctions_def group0.Group_ZF_2_1_L6
      group0.Group_ZF_3_1_L3 by simp
  ultimately show thesis
    using group0.group0_3_T3 by simp
qed
```

38.2 Almost homomorphisms

An almost homomorphism is a group valued function defined on a monoid M with the property that the set $\{f(m+n) - f(m) - f(n)\}_{m,n \in M}$ is finite. This term is used by R. D. Arthan in "The Eudoxus Real Numbers". We use this term in the general group context and use the A'Campo's term "slopes" (see his "A natural construction for the real numbers") to mean an almost homomorphism mapping interegers into themselves. We consider almost homomorphisms because we use slopes to define real numbers in the `Real_ZF_x` series.

HomDiff is an acronym for "homomorphism difference". This is the expression $s(mn)(s(m)s(n))^{-1}$, or $s(m+n) - s(m) - s(n)$ in the additive notation. It is equal to the neutral element of the group if s is a homomorphism.

```
definition
  HomDiff(G,f,s,x) ≡
  f⟨s(f⟨ fst(x),snd(x)⟩) ,
  (GroupInv(G,f)(f⟨ s(fst(x)),s(snd(x))⟩))⟩
```

Almost homomorphisms are defined as those maps $s : G \to G$ such that the homomorphism difference takes only finite number of values on $G \times G$.

```
definition
  AlmostHoms(G,f) ≡
  {s ∈ G→G.{HomDiff(G,f,s,x). x ∈ G×G } ∈ Fin(G)}
```

AlHomOp1(G, f) is the group operation on almost homomorphisms defined in a natural way by $(s \cdot r)(n) = s(n) \cdot r(n)$. In the terminology defined in func1.thy this is the group operation f (on G) lifted to the function space $G \to G$ and restricted to the set AlmostHoms(G, f).

```
definition
  AlHomOp1(G,f) ≡
  restrict(f {lifted to function space over} G,
  AlmostHoms(G,f)×AlmostHoms(G,f))
```

We also define a composition (binary) operator on almost homomorphisms in a natural way. We call that operator `AlHomOp2` - the second operation on almost homomorphisms. Composition of almost homomorphisms is used to define multiplication of real numbers in `Real_ZF` series.

```
definition
  AlHomOp2(G,f) ≡
  restrict(Composition(G),AlmostHoms(G,f)×AlmostHoms(G,f))
```

This lemma provides more readable notation for the HomDiff definition. Not really intended to be used in proofs, but just to see the definition in the notation defined in the group0 locale.

```
lemma (in group0) HomDiff_notation:
  shows HomDiff(G,P,s,⟨ m,n⟩) = s(m·n)·(s(m)·s(n))⁻¹
  using HomDiff_def by simp
```

The next lemma shows the set from the definition of almost homomorphism in a different form.

```
lemma (in group0) Group_ZF_3_2_L1A: shows
  {HomDiff(G,P,s,x). x ∈ G×G } = {s(m·n)·(s(m)·s(n))⁻¹. ⟨ m,n⟩ ∈ G×G}
proof -
  have ∀m∈G.∀n∈G. HomDiff(G,P,s,⟨ m,n⟩) = s(m·n)·(s(m)·s(n))⁻¹
    using HomDiff_notation by simp
  then show thesis by (rule ZF1_1_L4A)
qed
```

Let's define some notation. We inherit the notation and assumptions from the group0 context (locale) and add some. We will use AH to denote the set of almost homomorphisms. $\sim$ is the inverse (negative if the group is the group of integers) of almost homomorphisms, $(\sim p)(n) = p(n)^{-1}$. δ will denote the homomorphism difference specific for the group (HomDiff(G, f)). The notation $s \approx r$ will mean that s, r are almost equal, that is they are in the equivalence relation defined by the group of finite range functions (that is a normal subgroup of almost homomorphisms, if the group is abelian). We show that this is equivalent to the set $\{s(n) \cdot r(n)^{-1} : n \in G\}$ being finite. We also add an assumption that the G is abelian as many needed properties do not hold without that.

```
locale group1 = group0 +
  assumes isAbelian: P {is commutative on} G

  fixes AH
  defines AH_def [simp]: AH ≡ AlmostHoms(G,P)

  fixes Op1
  defines Op1_def [simp]: Op1 ≡ AlHomOp1(G,P)

  fixes Op2
```

```
  defines Op2_def [simp]: Op2 ≡ AlHomOp2(G,P)

  fixes FR
  defines FR_def [simp]: FR ≡ FinRangeFunctions(G,G)

  fixes neg (∼_ [90] 91)
  defines neg_def [simp]: ∼s ≡ GroupInv(G,P) O s

  fixes δ
  defines δ_def [simp]: δ(s,x) ≡ HomDiff(G,P,s,x)

  fixes AHprod (infix · 69)
  defines AHprod_def [simp]: s · r ≡  AlHomOp1(G,P)⟨s,r⟩

  fixes AHcomp (infix ∘ 70)
  defines AHcomp_def [simp]: s ∘ r ≡  AlHomOp2(G,P)⟨s,r⟩

  fixes AlEq (infix ≅ 68)
  defines AlEq_def [simp]: s ≅ r ≡ ⟨s,r⟩ ∈ QuotientGroupRel(AH,Op1,FR)
```

HomDiff is a homomorphism on the lifted group structure.

```
lemma (in group1) Group_ZF_3_2_L1:
  assumes A1: s:G→G  r:G→G
  and A2: x ∈ G×G
  and A3: F = P {lifted to function space over} G
  shows δ(F⟨ s,r⟩,x) = δ(s,x)·δ(r,x)
proof -
  let p = F⟨ s,r⟩
  from A2 obtain m n where
    D1: x = ⟨ m,n⟩ m∈G n∈G
    by auto
  then have T1:m·n ∈ G
    using group0_2_L1 monoid0.group0_1_L1 by simp
  with A1 D1 have T2:
    s(m)∈G s(n)∈G r(m)∈G
    r(n)∈G s(m·n)∈G r(m·n)∈G
    using apply_funtype by auto
  from A3 A1 have T3:p : G→G
    using group0_2_L1 monoid0.Group_ZF_2_1_L0
    by simp
  from D1 T3 have
    δ(p,x) = p(m·n)·((p(n))^-1·(p(m))^-1)
    using HomDiff_notation apply_funtype group_inv_of_two
    by simp
  also from A3 A1 D1 T1 isAbelian T2 have
    ... = δ(s,x)· δ(r,x)
    using Group_ZF_2_1_L3 group0_4_L3 HomDiff_notation
    by simp
  finally show thesis by simp
```

qed

The group operation lifted to the function space over G preserves almost homomorphisms.

```
lemma (in group1) Group_ZF_3_2_L2: assumes A1: s ∈ AH r ∈ AH
  and A2: F = P {lifted to function space over} G
  shows F⟨ s,r⟩ ∈ AH
proof -
  let p = F⟨ s,r⟩
  from A1 A2 have p : G→G
    using AlmostHoms_def group0_2_L1 monoid0.Group_ZF_2_1_L0
    by simp
  moreover have
    {δ(p,x). x ∈ G×G} ∈ Fin(G)
  proof -
    from A1 have
      {δ(s,x). x ∈ G×G } ∈ Fin(G)
      {δ(r,x). x ∈ G×G } ∈ Fin(G)
      using AlmostHoms_def by auto
    with groupAssum A1 A2 show thesis
      using IsAgroup_def IsAmonoid_def IsAssociative_def
      Finite1_L15 AlmostHoms_def Group_ZF_3_2_L1
      by auto
  qed
  ultimately show thesis using AlmostHoms_def
    by simp
qed
```

The set of almost homomorphisms is closed under the lifted group operation.

```
lemma (in group1) Group_ZF_3_2_L3:
  assumes F = P {lifted to function space over} G
  shows AH {is closed under} F
  using assms IsOpClosed_def Group_ZF_3_2_L2 by simp
```

The terms in the homomorphism difference for a function are in the group.

```
lemma (in group1) Group_ZF_3_2_L4:
  assumes s:G→G and m∈G  n∈G
  shows
  m·n ∈ G
  s(m·n) ∈ G
  s(m) ∈ G s(n) ∈ G
  δ(s,⟨ m,n⟩) ∈ G
  s(m)·s(n) ∈ G
  using assms group_op_closed inverse_in_group
    apply_funtype HomDiff_def by auto
```

It is handy to have a version of `Group_ZF_3_2_L4` specifically for almost homomorphisms.

```
corollary (in group1) Group_ZF_3_2_L4A:
  assumes s ∈ AH and m∈G  n∈G
  shows m·n ∈ G
  s(m·n) ∈ G
  s(m) ∈ G s(n) ∈ G
  δ(s,⟨ m,n⟩) ∈ G
  s(m)·s(n) ∈ G
  using assms AlmostHoms_def Group_ZF_3_2_L4
  by auto
```

The terms in the homomorphism difference are in the group, a different form.

```
lemma (in group1) Group_ZF_3_2_L4B:
  assumes A1:s ∈ AH and A2:x∈G×G
  shows fst(x)·snd(x) ∈ G
  s(fst(x)·snd(x)) ∈ G
  s(fst(x)) ∈ G s(snd(x)) ∈ G
  δ(s,x) ∈ G
  s(fst(x))·s(snd(x)) ∈ G
proof -
  let m = fst(x)
  let n = snd(x)
  from A1 A2 show
    m·n ∈ G  s(m·n) ∈ G
    s(m) ∈ G s(n) ∈ G
    s(m)·s(n) ∈ G
    using Group_ZF_3_2_L4A
    by auto
  from A1 A2 have δ(s,⟨ m,n⟩) ∈ G using Group_ZF_3_2_L4A
    by simp
  moreover from A2 have ⟨ m,n⟩ = x by auto
  ultimately show δ(s,x) ∈ G by simp
qed
```

What are the values of the inverse of an almost homomorphism?

```
lemma (in group1) Group_ZF_3_2_L5:
  assumes s ∈ AH and n∈G
  shows (∼s)(n) = (s(n))⁻¹
  using assms AlmostHoms_def comp_fun_apply by auto
```

Homomorphism difference commutes with the inverse for almost homomorphisms.

```
lemma (in group1) Group_ZF_3_2_L6:
  assumes A1:s ∈ AH and A2:x∈G×G
  shows δ(∼s,x) = (δ(s,x))⁻¹
proof -
  let m = fst(x)
  let n = snd(x)
```

```
  have δ(∼s,x) = (∼s)(m·n)·((∼s)(m)·(∼s)(n))⁻¹
    using HomDiff_def by simp
  from A1 A2 isAbelian show thesis
    using Group_ZF_3_2_L4B HomDiff_def
      Group_ZF_3_2_L5 group0_4_L4A
    by simp
qed
```

The inverse of an almost homomorphism maps the group into itself.

```
lemma (in group1) Group_ZF_3_2_L7:
  assumes s ∈ AH
  shows ∼s : G→G
  using groupAssum assms AlmostHoms_def group0_2_T2 comp_fun by auto
```

The inverse of an almost homomorphism is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_2_L8:
  assumes A1: F = P {lifted to function space over} G
  and A2: s ∈ AH
  shows GroupInv(G→G,F)(s) ∈ AH
proof -
  from A2 have {δ(s,x). x ∈ G×G} ∈ Fin(G)
    using AlmostHoms_def by simp
  with groupAssum  have
    GroupInv(G,P){δ(s,x). x ∈ G×G} ∈ Fin(G)
    using group0_2_T2 Finite1_L6A by blast
  moreover have
     GroupInv(G,P){δ(s,x). x ∈ G×G} =
    {(δ(s,x))⁻¹. x ∈ G×G}
  proof -
    from groupAssum have
      GroupInv(G,P) : G→G
      using group0_2_T2 by simp
    moreover from A2 have
      ∀x∈G×G. δ(s,x)∈G
      using Group_ZF_3_2_L4B by simp
    ultimately show thesis
      using func1_1_L17 by simp
  qed
  ultimately have {(δ(s,x))⁻¹. x ∈ G×G} ∈ Fin(G)
    by simp
  moreover from A2 have
    {(δ(s,x))⁻¹. x ∈ G×G} = {δ(∼s,x). x ∈ G×G}
    using Group_ZF_3_2_L6 by simp
  ultimately have {δ(∼s,x). x ∈ G×G} ∈ Fin(G)
    by simp
  with A2 groupAssum A1 show thesis
    using Group_ZF_3_2_L7 AlmostHoms_def Group_ZF_2_1_L6
    by simp
qed
```

The function that assigns the neutral element everywhere is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_2_L9: shows
  ConstantFunction(G,1) ∈ AH and AH≠0
proof -
  let z = ConstantFunction(G,1)
  have G×G≠0 using group0_2_L1 monoid0.group0_1_L3A
    by blast
  moreover have ∀x∈G×G. δ(z,x) = 1
  proof
    fix x assume A1:x ∈ G × G
    then obtain m n where x = ⟨ m,n⟩ m∈G n∈G
      by auto
    then show δ(z,x) = 1
      using group0_2_L1 monoid0.group0_1_L1
 func1_3_L2 HomDiff_def group0_2_L2
 group_inv_of_one by simp
  qed
  ultimately have {δ(z,x). x∈G×G} = {1} by (rule ZF1_1_L5)
  then show z ∈ AH using group0_2_L2 Finite1_L16
    func1_3_L1 group0_2_L2 AlmostHoms_def by simp
  then show AH≠0 by auto
qed
```

If the group is abelian, then almost homomorphisms form a subgroup of the lifted group.

```
lemma Group_ZF_3_2_L10:
  assumes A1: IsAgroup(G,P)
  and A2: P {is commutative on} G
  and A3: F = P {lifted to function space over} G
  shows IsAsubgroup(AlmostHoms(G,P),F)
proof -
  let AH = AlmostHoms(G,P)
  from A2 A1 have T1: group1(G,P)
    using group1_axioms.intro group0_def group1_def
    by simp
  from A1 A3 have group0(G→G,F)
    using group0_def group0.Group_ZF_2_1_T2 by simp
  moreover from T1 have AH≠0
    using group1.Group_ZF_3_2_L9 by simp
  moreover have T2:AH ⊆ G→G
    using AlmostHoms_def by auto
  moreover from T1 A3 have
    AH {is closed under} F
    using group1.Group_ZF_3_2_L3 by simp
  moreover from T1 A3 have
    ∀s∈AH. GroupInv(G→G,F)(s) ∈ AH
    using group1.Group_ZF_3_2_L8 by simp
  ultimately show IsAsubgroup(AlmostHoms(G,P),F)
```

```
    using group0.group0_3_T3 by simp
qed
```

If the group is abelian, then almost homomorphisms form a group with the first operation, hence we can use theorems proven in group0 context aplied to this group.

```
lemma (in group1) Group_ZF_3_2_L10A:
  shows IsAgroup(AH,Op1) group0(AH,Op1)
    using groupAssum isAbelian Group_ZF_3_2_L10 IsAsubgroup_def
      AlHomOp1_def group0_def by auto
```

The group of almost homomorphisms is abelian

```
lemma Group_ZF_3_2_L11: assumes A1: IsAgroup(G,f)
  and A2: f {is commutative on} G
  shows
  IsAgroup(AlmostHoms(G,f),AlHomOp1(G,f))
  AlHomOp1(G,f) {is commutative on} AlmostHoms(G,f)
proof-
  let AH = AlmostHoms(G,f)
  let F = f {lifted to function space over} G
  from A1 A2 have IsAsubgroup(AH,F)
    using Group_ZF_3_2_L10 by simp
  then show IsAgroup(AH,AlHomOp1(G,f))
    using IsAsubgroup_def AlHomOp1_def by simp
  from A1 have F : (G→G)×(G→G)→(G→G)
    using IsAgroup_def monoid0_def monoid0.Group_ZF_2_1_L0A
    by simp
  moreover have AH ⊆ G→G
    using AlmostHoms_def by auto
  moreover from A1 A2 have
    F {is commutative on} (G→G)
    using group0_def group0.Group_ZF_2_1_L7
    by simp
  ultimately show
    AlHomOp1(G,f){is commutative on} AH
    using func_ZF_4_L1 AlHomOp1_def by simp
qed
```

The first operation on homomorphisms acts in a natural way on its operands.

```
lemma (in group1) Group_ZF_3_2_L12:
  assumes s∈AH  r∈AH and n∈G
  shows (s·r)(n) = s(n)·r(n)
  using assms AlHomOp1_def restrict AlmostHoms_def Group_ZF_2_1_L3
  by simp
```

What is the group inverse in the group of almost homomorphisms?

```
lemma (in group1) Group_ZF_3_2_L13:
  assumes A1: s∈AH
```

```
  shows
  GroupInv(AH,Op1)(s) = GroupInv(G,P) O s
  GroupInv(AH,Op1)(s) ∈ AH
  GroupInv(G,P) O s ∈ AH
proof -
  let F = P {lifted to function space over} G
  from groupAssum isAbelian have IsAsubgroup(AH,F)
    using Group_ZF_3_2_L10 by simp
  with A1 show I: GroupInv(AH,Op1)(s) = GroupInv(G,P) O s
    using AlHomOp1_def Group_ZF_2_1_L6A by simp
  from A1 show GroupInv(AH,Op1)(s) ∈ AH
    using Group_ZF_3_2_L10A group0.inverse_in_group by simp
  with I show GroupInv(G,P) O s ∈ AH by simp
qed
```

The group inverse in the group of almost homomorphisms acts in a natural way on its operand.

```
lemma (in group1) Group_ZF_3_2_L14:
  assumes s∈AH and n∈G
  shows (GroupInv(AH,Op1)(s))(n) = (s(n))⁻¹
  using isAbelian assms Group_ZF_3_2_L13 AlmostHoms_def comp_fun_apply
  by auto
```

The next lemma states that if s, r are almost homomorphisms, then $s \cdot r^{-1}$ is also an almost homomorphism.

```
lemma Group_ZF_3_2_L15: assumes IsAgroup(G,f)
  and f {is commutative on} G
  and AH = AlmostHoms(G,f) Op1 = AlHomOp1(G,f)
  and s ∈ AH  r ∈ AH
  shows
  Op1⟨ s,r⟩ ∈ AH
  GroupInv(AH,Op1)(r) ∈ AH
  Op1⟨ s,GroupInv(AH,Op1)(r)⟩ ∈ AH
  using assms group0_def group1_axioms.intro group1_def
      group1.Group_ZF_3_2_L10A group0.group0_2_L1
      monoid0.group0_1_L1 group0.inverse_in_group by auto
```

A version of `Group_ZF_3_2_L15` formulated in notation used in `group1` context. States that the product of almost homomorphisms is an almost homomorphism and the the product of an almost homomorphism with a (pointwise) inverse of an almost homomorphism is an almost homomorphism.

```
corollary (in group1) Group_ZF_3_2_L16: assumes s ∈ AH  r ∈ AH
  shows s·r ∈ AH  s·(∼r) ∈ AH
  using assms isAbelian group0_def group1_axioms group1_def
  Group_ZF_3_2_L15 Group_ZF_3_2_L13 by auto
```

38.3 The classes of almost homomorphisms

In the Real_ZF series we define real numbers as a quotient of the group of integer almost homomorphisms by the integer finite range functions. In this section we setup the background for that in the general group context.

Finite range functions are almost homomorphisms.

```
lemma (in group1) Group_ZF_3_3_L1: shows FR ⊆ AH
proof
  fix s assume A1:s ∈ FR
  then have T1:{s(n). n ∈ G} ∈ Fin(G)
    {s(fst(x)). x∈G×G} ∈ Fin(G)
    {s(snd(x)). x∈G×G} ∈ Fin(G)
    using Finite1_L18 Finite1_L6B by auto
  have {s(fst(x)·snd(x)). x ∈ G×G} ∈ Fin(G)
  proof -
    have ∀x∈G×G. fst(x)·snd(x) ∈ G
      using group0_2_L1 monoid0.group0_1_L1 by simp
    moreover from T1 have {s(n). n ∈ G} ∈ Fin(G) by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  moreover have
    {(s(fst(x))·s(snd(x)))⁻¹. x∈G×G} ∈ Fin(G)
  proof -
    have ∀g∈G. g⁻¹ ∈ G using inverse_in_group
      by simp
    moreover from T1 have
      {s(fst(x))·s(snd(x)). x∈G×G} ∈ Fin(G)
      using group_oper_fun  Finite1_L15 by simp
    ultimately show thesis
      by (rule Finite1_L6C)
  qed
  ultimately have {δ(s,x). x∈G×G} ∈ Fin(G)
    using HomDiff_def Finite1_L15  group_oper_fun
    by simp
  with A1 show s ∈ AH
    using FinRangeFunctions_def AlmostHoms_def
    by simp
qed
```

Finite range functions valued in an abelian group form a normal subgroup of almost homomorphisms.

```
lemma Group_ZF_3_3_L2: assumes A1:IsAgroup(G,f)
  and A2:f {is commutative on} G
  shows
  IsAsubgroup(FinRangeFunctions(G,G),AlHomOp1(G,f))
  IsAnormalSubgroup(AlmostHoms(G,f),AlHomOp1(G,f),
  FinRangeFunctions(G,G))
proof -
```

```
  let H1 = AlmostHoms(G,f)
  let H2 = FinRangeFunctions(G,G)
  let F = f {lifted to function space over} G
  from A1 A2 have T1:group0(G,f)
    monoid0(G,f) group1(G,f)
    using group0_def group0.group0_2_L1
      group1_axioms.intro group1_def
    by auto
  with A1 A2 have IsAgroup(G→G,F)
    IsAsubgroup(H1,F) IsAsubgroup(H2,F)
    using group0.Group_ZF_2_1_T2 Group_ZF_3_2_L10
      monoid0.group0_1_L3A Group_ZF_3_1_T1
    by auto
  then have
    IsAsubgroup(H1∩H2,restrict(F,H1×H1))
    using group0_3_L7 by simp
  moreover from T1 have H1∩H2 = H2
    using group1.Group_ZF_3_3_L1 by auto
  ultimately show IsAsubgroup(H2,AlHomOp1(G,f))
    using AlHomOp1_def by simp
  with A1 A2 show IsAnormalSubgroup(AlmostHoms(G,f),AlHomOp1(G,f),
    FinRangeFunctions(G,G))
    using Group_ZF_3_2_L11 Group_ZF_2_4_L6
    by simp
qed
```

The group of almost homomorphisms divided by the subgroup of finite range functions is an abelian group.

```
theorem (in group1) Group_ZF_3_3_T1:
  shows
  IsAgroup(AH//QuotientGroupRel(AH,Op1,FR),QuotientGroupOp(AH,Op1,FR))
  and
  QuotientGroupOp(AH,Op1,FR) {is commutative on}
  (AH//QuotientGroupRel(AH,Op1,FR))
  using groupAssum isAbelian Group_ZF_3_3_L2 Group_ZF_3_2_L10A
    Group_ZF_2_4_T1 Group_ZF_3_2_L10A Group_ZF_3_2_L11
    Group_ZF_3_3_L2 IsAnormalSubgroup_def Group_ZF_2_4_L6 by auto
```

It is useful to have a direct statement that the quotient group relation is an equivalence relation for the group of AH and subgroup FR.

```
lemma (in group1) Group_ZF_3_3_L3: shows
  QuotientGroupRel(AH,Op1,FR) ⊆ AH × AH and
  equiv(AH,QuotientGroupRel(AH,Op1,FR))
  using groupAssum isAbelian QuotientGroupRel_def
    Group_ZF_3_3_L2 Group_ZF_3_2_L10A group0.Group_ZF_2_4_L3
  by auto
```

The ”almost equal” relation is symmetric.

```
lemma (in group1) Group_ZF_3_3_L3A: assumes A1: s≅r
```

```
  shows r≅s
proof -
  let R = QuotientGroupRel(AH,Op1,FR)
  from A1 have equiv(AH,R) and ⟨s,r⟩ ∈ R
    using Group_ZF_3_3_L3 by simp_all
  then have ⟨r,s⟩ ∈ R by (rule equiv_is_sym)
  then show r≅s by simp
qed
```

Although we have bypassed this fact when proving that group of almost homomorphisms divided by the subgroup of finite range functions is a group, it is still useful to know directly that the first group operation on AH is congruent with respect to the quotient group relation.

```
lemma (in group1) Group_ZF_3_3_L4:
  shows Congruent2(QuotientGroupRel(AH,Op1,FR),Op1)
  using groupAssum isAbelian Group_ZF_3_2_L10A Group_ZF_3_3_L2
    Group_ZF_2_4_L5A by simp
```

The class of an almost homomorphism s is the neutral element of the quotient group of almost homomorphisms iff s is a finite range function.

```
lemma (in group1) Group_ZF_3_3_L5: assumes s ∈ AH and
  r = QuotientGroupRel(AH,Op1,FR) and
  TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR)) = e
  shows r{s} = e ⟷ s ∈ FR
  using groupAssum isAbelian assms Group_ZF_3_2_L11
    group0_def Group_ZF_3_3_L2 group0.Group_ZF_2_4_L5E
  by simp
```

The group inverse of a class of an almost homomorphism f is the class of the inverse of f.

```
lemma (in group1) Group_ZF_3_3_L6:
  assumes A1: s ∈ AH  and
  r = QuotientGroupRel(AH,Op1,FR) and
  F = ProjFun2(AH,r,Op1)
  shows r{∼s} = GroupInv(AH//r,F)(r{s})
proof -
  from groupAssum isAbelian assms have
    r{GroupInv(AH, Op1)(s)} = GroupInv(AH//r,F)(r  {s})
    using Group_ZF_3_2_L10A Group_ZF_3_3_L2 QuotientGroupOp_def
      group0.Group_ZF_2_4_L7 by simp
  with A1 show thesis using Group_ZF_3_2_L13
    by simp
qed
```

38.4 Compositions of almost homomorphisms

The goal of this section is to establish some facts about composition of almost homomorphisms that are needed for the real numbers construction in `Real_ZF_x` series. In particular we show that the set of almost homomorphisms is closed under composition and that composition is congruent with respect to the equivalence relation defined by the group of finite range functions (a normal subgroup of almost homomorphisms).

The next formula restates the definition of the homomorphism difference to express the value an almost homomorphism on a product.

```
lemma (in group1) Group_ZF_3_4_L1:
  assumes s∈AH and  m∈G  n∈G
  shows s(m·n) = s(m)·s(n)·δ(s,⟨ m,n⟩)
  using isAbelian assms Group_ZF_3_2_L4A HomDiff_def group0_4_L5
  by simp
```

What is the value of a composition of almost homomorhisms?

```
lemma (in group1) Group_ZF_3_4_L2:
  assumes s∈AH  r∈AH and m∈G
  shows (s∘r)(m) = s(r(m))  s(r(m)) ∈ G
  using assms AlmostHoms_def func_ZF_5_L3 restrict AlHomOp2_def
    apply_funtype by auto
```

What is the homomorphism difference of a composition?

```
lemma (in group1) Group_ZF_3_4_L3:
  assumes A1: s∈AH  r∈AH and A2: m∈G  n∈G
  shows δ(s∘r,⟨ m,n⟩) =
  δ(s,⟨ r(m),r(n)⟩)·s(δ(r,⟨ m,n⟩))·δ(s,⟨ r(m)·r(n),δ(r,⟨ m,n⟩)⟩)
proof -
  from A1 A2 have T1:
    s(r(m))· s(r(n)) ∈ G
    δ(s,⟨ r(m),r(n)⟩)∈ G s(δ(r,⟨ m,n⟩)) ∈G
    δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩) ∈ G
    using Group_ZF_3_4_L2 AlmostHoms_def apply_funtype
      Group_ZF_3_2_L4A group0_2_L1 monoid0.group0_1_L1
    by auto
  from A1 A2 have δ(s∘r,⟨ m,n⟩) =
    s(r(m)·r(n)·δ(r,⟨ m,n⟩))·(s((r(m)))·s(r(n)))⁻¹
    using HomDiff_def group0_2_L1 monoid0.group0_1_L1 Group_ZF_3_4_L2
      Group_ZF_3_4_L1 by simp
  moreover from A1 A2 have
    s(r(m)·r(n)·δ(r,⟨ m,n⟩)) =
    s(r(m)·r(n))·s(δ(r,⟨ m,n⟩))·δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)
    s(r(m)·r(n)) = s(r(m))·s(r(n))·δ(s,⟨ r(m),r(n)⟩)
    using Group_ZF_3_2_L4A Group_ZF_3_4_L1 by auto
  moreover from T1 isAbelian have
    s(r(m))·s(r(n))·δ(s,⟨ r(m),r(n)⟩)·
    s(δ(r,⟨ m,n⟩))·δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)·
    (s((r(m)))·s(r(n)))⁻¹ =
    δ(s,⟨ r(m),r(n)⟩)·s(δ(r,⟨ m,n⟩))·δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)
    using group0_4_L6C by simp
```

```
  ultimately show thesis by simp
qed
```

What is the homomorphism difference of a composition (another form)? Here we split the homomorphism difference of a composition into a product of three factors. This will help us in proving that the range of homomorphism difference for the composition is finite, as each factor has finite range.

```
lemma (in group1) Group_ZF_3_4_L4:
  assumes A1: s∈AH  r∈AH and A2: x ∈ G×G
  and A3:
  A = δ(s,⟨ r(fst(x)),r(snd(x))⟩)
  B = s(δ(r,x))
  C = δ(s,⟨ (r(fst(x))·r(snd(x))),δ(r,x)⟩)
  shows δ(s∘r,x) = A·B·C
proof -
  let m = fst(x)
  let n = snd(x)
  note A1
  moreover from A2 have m∈G n∈G
    by auto
  ultimately have
    δ(s∘r,⟨ m,n⟩) =
    δ(s,⟨ r(m),r(n)⟩)·s(δ(r,⟨ m,n⟩))·
    δ(s,⟨ (r(m)·r(n)),δ(r,⟨ m,n⟩)⟩)
    by (rule Group_ZF_3_4_L3)
  with A1 A2 A3 show thesis
    by auto
qed
```

The range of the homomorphism difference of a composition of two almost homomorphisms is finite. This is the essential condition to show that a composition of almost homomorphisms is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_4_L5:
  assumes A1: s∈AH  r∈AH
  shows {δ(Composition(G)⟨ s,r⟩,x). x ∈ G×G} ∈ Fin(G)
proof -
  from A1 have
    ∀x∈G×G. ⟨ r(fst(x)),r(snd(x))⟩ ∈ G×G
    using Group_ZF_3_2_L4B by simp
  moreover from A1 have
    {δ(s,x). x∈G×G} ∈ Fin(G)
    using AlmostHoms_def by simp
  ultimately have
    {δ(s,⟨ r(fst(x)),r(snd(x))⟩). x∈G×G} ∈ Fin(G)
    by (rule Finite1_L6B)
  moreover have {s(δ(r,x)). x∈G×G} ∈ Fin(G)
  proof -
    from A1 have ∀m∈G. s(m) ∈ G
```

```
      using AlmostHoms_def apply_funtype by auto
    moreover from A1 have {δ(r,x). x∈G×G} ∈ Fin(G)
      using AlmostHoms_def by simp
    ultimately show thesis
      by (rule Finite1_L6C)
  qed
  ultimately have
    {δ(s,⟨ r(fst(x)),r(snd(x))⟩)·s(δ(r,x)). x∈G×G} ∈ Fin(G)
    using group_oper_fun Finite1_L15 by simp
  moreover have
    {δ(s,⟨ (r(fst(x))·r(snd(x))),δ(r,x)⟩).  x∈G×G} ∈ Fin(G)
  proof -
    from A1 have
    ∀x∈G×G. ⟨ (r(fst(x))·r(snd(x))),δ(r,x)⟩ ∈ G×G
      using Group_ZF_3_2_L4B by simp
    moreover from A1 have
      {δ(s,x). x∈G×G} ∈ Fin(G)
      using AlmostHoms_def by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  ultimately have
    {δ(s,⟨ r(fst(x)),r(snd(x))⟩)·s(δ(r,x))·
    δ(s,⟨ (r(fst(x))·r(snd(x))),δ(r,x)⟩). x∈G×G} ∈ Fin(G)
    using group_oper_fun Finite1_L15 by simp
  moreover from A1 have {δ(s∘r,x). x∈G×G} =
    {δ(s,⟨ r(fst(x)),r(snd(x))⟩)·s(δ(r,x))·
    δ(s,⟨ (r(fst(x))·r(snd(x))),δ(r,x)⟩). x∈G×G}
    using Group_ZF_3_4_L4 by simp
  ultimately have {δ(s∘r,x). x∈G×G} ∈ Fin(G) by simp
  with A1 show thesis using restrict AlHomOp2_def
    by simp
qed
```

Composition of almost homomorphisms is an almost homomorphism.

```
theorem (in group1) Group_ZF_3_4_T1:
  assumes A1: s∈AH  r∈AH
  shows Composition(G)⟨ s,r⟩ ∈ AH s∘r ∈ AH
proof -
  from A1 have ⟨ s,r⟩ ∈ (G→G)×(G→G)
    using AlmostHoms_def by simp
  then have Composition(G)⟨ s,r⟩ : G→G
    using func_ZF_5_L1 apply_funtype by blast
  with A1 show Composition(G)⟨ s,r⟩ ∈ AH
    using Group_ZF_3_4_L5 AlmostHoms_def
    by simp
  with A1 show  s∘r ∈ AH using AlHomOp2_def restrict
    by simp
qed
```

The set of almost homomorphisms is closed under composition. The second operation on almost homomorphisms is associative.

```
lemma (in group1) Group_ZF_3_4_L6: shows
  AH {is closed under} Composition(G)
  AlHomOp2(G,P) {is associative on} AH
proof -
  show AH {is closed under} Composition(G)
    using Group_ZF_3_4_T1 IsOpClosed_def by simp
  moreover have AH ⊆ G→G using AlmostHoms_def
    by auto
  moreover have
    Composition(G) {is associative on} (G→G)
    using func_ZF_5_L5 by simp
  ultimately show AlHomOp2(G,P) {is associative on} AH
    using func_ZF_4_L3 AlHomOp2_def by simp
qed
```

Type information related to the situation of two almost homomorphisms.

```
lemma (in group1) Group_ZF_3_4_L7:
  assumes A1: s∈AH  r∈AH and A2: n∈G
  shows
  s(n) ∈ G (r(n))⁻¹ ∈ G
  s(n)·(r(n))⁻¹ ∈ G   s(r(n)) ∈ G
proof -
  from A1 A2 show
    s(n) ∈ G
    (r(n))⁻¹ ∈ G
    s(r(n)) ∈ G
    s(n)·(r(n))⁻¹ ∈ G
    using AlmostHoms_def apply_type
      group0_2_L1 monoid0.group0_1_L1 inverse_in_group
    by auto
qed
```

Type information related to the situation of three almost homomorphisms.

```
lemma (in group1) Group_ZF_3_4_L8:
  assumes A1: s∈AH  r∈AH  q∈AH and A2: n∈G
  shows
  q(n)∈G
  s(r(n)) ∈ G
  r(n)·(q(n))⁻¹ ∈ G
  s(r(n)·(q(n))⁻¹) ∈ G
  δ(s,⟨ q(n),r(n)·(q(n))⁻¹⟩) ∈ G
proof -
  from A1 A2 show
    q(n)∈ G  s(r(n)) ∈ G r(n)·(q(n))⁻¹ ∈ G
    using AlmostHoms_def apply_type
      group0_2_L1 monoid0.group0_1_L1 inverse_in_group
    by auto
  with A1 A2 show s(r(n)·(q(n))⁻¹) ∈ G
```

```
    δ(s,⟨ q(n),r(n)·(q(n))⁻¹⟩) ∈ G
    using AlmostHoms_def apply_type Group_ZF_3_2_L4A
    by auto
qed
```

A formula useful in showing that the composition of almost homomorphisms is congruent with respect to the quotient group relation.

```
lemma (in group1) Group_ZF_3_4_L9:
  assumes A1: s1 ∈ AH  r1 ∈ AH  s2 ∈ AH  r2 ∈ AH
  and A2: n∈G
  shows (s1∘r1)(n)·((s2∘r2)(n))⁻¹ =
  s1(r2(n))· (s2(r2(n)))⁻¹·s1(r1(n)·(r2(n))⁻¹)·
  δ(s1,⟨ r2(n),r1(n)·(r2(n))⁻¹⟩)
proof -
  from A1 A2 isAbelian have
    (s1∘r1)(n)·((s2∘r2)(n))⁻¹ =
    s1(r2(n)·(r1(n)·(r2(n))⁻¹))·(s2(r2(n)))⁻¹
    using Group_ZF_3_4_L2 Group_ZF_3_4_L7 group0_4_L6A
      group_oper_assoc by simp
  with A1 A2 have (s1∘r1)(n)·((s2∘r2)(n))⁻¹ = s1(r2(n))·
    s1(r1(n)·(r2(n))⁻¹)·δ(s1,⟨ r2(n),r1(n)·(r2(n))⁻¹⟩)·
    (s2(r2(n)))⁻¹
    using Group_ZF_3_4_L8 Group_ZF_3_4_L1 by simp
  with A1 A2 isAbelian show thesis using
    Group_ZF_3_4_L8 group0_4_L7 by simp
qed
```

The next lemma shows a formula that translates an expression in terms of the first group operation on almost homomorphisms and the group inverse in the group of almost homomorphisms to an expression using only the underlying group operations.

```
lemma (in group1) Group_ZF_3_4_L10: assumes A1: s ∈ AH  r ∈ AH
  and A2: n ∈ G
  shows (s·(GroupInv(AH,Op1)(r)))(n) = s(n)·(r(n))⁻¹
proof -
  from A1 A2 show thesis
    using isAbelian Group_ZF_3_2_L13 Group_ZF_3_2_L12 Group_ZF_3_2_L14
    by simp
qed
```

A neccessary condition for two a. h. to be almost equal.

```
lemma (in group1) Group_ZF_3_4_L11:
  assumes A1: s≅r
  shows {s(n)·(r(n))⁻¹. n∈G} ∈ Fin(G)
proof -
  from A1 have s∈AH r∈AH
    using QuotientGroupRel_def by auto
  moreover from A1 have
```

```
    {(s·(GroupInv(AH,Op1)(r)))(n). n∈G} ∈ Fin(G)
    using QuotientGroupRel_def Finite1_L18 by simp
  ultimately show thesis
    using Group_ZF_3_4_L10 by simp
qed
```

A sufficient condition for two a. h. to be almost equal.

```
lemma (in group1) Group_ZF_3_4_L12: assumes A1: s∈AH  r∈AH
  and A2: {s(n)·(r(n))⁻¹. n∈G} ∈ Fin(G)
  shows s≅r
proof -
  from groupAssum isAbelian A1 A2 show thesis
    using Group_ZF_3_2_L15 AlmostHoms_def
    Group_ZF_3_4_L10 Finite1_L19 QuotientGroupRel_def
    by simp
qed
```

Another sufficient consdition for two a.h. to be almost equal. It is actually just an expansion of the definition of the quotient group relation.

```
lemma (in group1) Group_ZF_3_4_L12A: assumes s∈AH  r∈AH
  and s·(GroupInv(AH,Op1)(r)) ∈ FR
  shows s≅r  r≅s
proof  -
  from assms show s≅r using assms QuotientGroupRel_def
    by simp
  then show r≅s by (rule Group_ZF_3_3_L3A)
qed
```

Another necessary condition for two a.h. to be almost equal. It is actually just an expansion of the definition of the quotient group relation.

```
lemma (in group1) Group_ZF_3_4_L12B: assumes s≅r
  shows s·(GroupInv(AH,Op1)(r)) ∈ FR
  using assms QuotientGroupRel_def by simp
```

The next lemma states the essential condition for the composition of a. h. to be congruent with respect to the quotient group relation for the subgroup of finite range functions.

```
lemma (in group1) Group_ZF_3_4_L13:
  assumes A1: s1≅s2  r1≅r2
  shows (s1∘r1) ≅ (s2∘r2)
proof -
  have {s1(r2(n))· (s2(r2(n)))⁻¹. n∈G} ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. r2(n) ∈ G
      using QuotientGroupRel_def AlmostHoms_def apply_funtype
      by auto
    moreover from A1 have {s1(n)·(s2(n))⁻¹. n∈G} ∈ Fin(G)
      using Group_ZF_3_4_L11 by simp
```

```
    ultimately show thesis by (rule Finite1_L6B)
  qed
  moreover have {s1(r1(n)·(r2(n))⁻¹). n ∈ G} ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. s1(n)∈G
      using QuotientGroupRel_def AlmostHoms_def apply_funtype
      by auto
    moreover from A1 have {r1(n)·(r2(n))⁻¹. n∈G} ∈ Fin(G)
      using Group_ZF_3_4_L11 by simp
    ultimately show thesis by (rule Finite1_L6C)
  qed
  ultimately have
    {s1(r2(n))· (s2(r2(n)))⁻¹·s1(r1(n)·(r2(n))⁻¹).
    n∈G} ∈ Fin(G)
    using group_oper_fun Finite1_L15 by simp
  moreover have
    {δ(s1,⟨ r2(n),r1(n)·(r2(n))⁻¹⟩). n∈G} ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. ⟨ r2(n),r1(n)·(r2(n))⁻¹⟩ ∈ G×G
      using QuotientGroupRel_def Group_ZF_3_4_L7 by auto
    moreover from A1 have {δ(s1,x). x ∈ G×G} ∈ Fin(G)
      using QuotientGroupRel_def AlmostHoms_def by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  ultimately have
    {s1(r2(n))· (s2(r2(n)))⁻¹·s1(r1(n)·(r2(n))⁻¹)·
    δ(s1,⟨ r2(n),r1(n)·(r2(n))⁻¹⟩). n∈G} ∈ Fin(G)
    using group_oper_fun Finite1_L15 by simp
  with A1 show thesis using
    QuotientGroupRel_def Group_ZF_3_4_L9
    Group_ZF_3_4_T1 Group_ZF_3_4_L12 by simp
qed
```

Composition of a. h. to is congruent with respect to the quotient group relation for the subgroup of finite range functions. Recall that if an operation say "∘" on X is congruent with respect to an equivalence relation R then we can define the operation on the quotient space X/R by $[s]_R \circ [r]_R := [s \circ r]_R$ and this definition will be correct i.e. it will not depend on the choice of representants for the classes $[x]$ and $[y]$. This is why we want it here.

```
lemma (in group1) Group_ZF_3_4_L13A: shows
  Congruent2(QuotientGroupRel(AH,Op1,FR),Op2)
proof -
  show thesis using Group_ZF_3_4_L13 Congruent2_def
    by simp
qed
```

The homomorphism difference for the identity function is equal to the neutral element of the group (denoted e in the group1 context).

```
lemma (in group1) Group_ZF_3_4_L14: assumes A1: x ∈ G×G
  shows δ(id(G),x) = 1
proof -
  from A1 show thesis using
    group0_2_L1 monoid0.group0_1_L1 HomDiff_def id_conv group0_2_L6
    by simp
qed
```

The identity function ($I(x) = x$) on G is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_4_L15: shows id(G) ∈ AH
proof -
  have G×G ≠ 0 using group0_2_L1 monoid0.group0_1_L3A
    by blast
  then show thesis using Group_ZF_3_4_L14 group0_2_L2
    id_type AlmostHoms_def by simp
qed
```

Almost homomorphisms form a monoid with composition. The identity function on the group is the neutral element there.

```
lemma (in group1) Group_ZF_3_4_L16:
  shows
  IsAmonoid(AH,Op2)
  monoid0(AH,Op2)
  id(G) = TheNeutralElement(AH,Op2)
proof-
  let i = TheNeutralElement(G→G,Composition(G))
  have
    IsAmonoid(G→G,Composition(G))
    monoid0(G→G,Composition(G))
    using monoid0_def Group_ZF_2_5_L2 by auto
  moreover have AH {is closed under} Composition(G)
    using Group_ZF_3_4_L6 by simp
  moreover have AH ⊆ G→G
    using AlmostHoms_def by auto
  moreover have i ∈ AH
    using Group_ZF_2_5_L2 Group_ZF_3_4_L15 by simp
  moreover have id(G) =  i
    using Group_ZF_2_5_L2 by simp
  ultimately show
    IsAmonoid(AH,Op2)
    monoid0(AH,Op2)
    id(G) = TheNeutralElement(AH,Op2)
    using monoid0.group0_1_T1 group0_1_L6 AlHomOp2_def monoid0_def
    by auto
qed
```

We can project the monoid of almost homomorphisms with composition to the group of almost homomorphisms divided by the subgroup of finite range functions. The class of the identity function is the neutral element of the quotient (monoid).

```
theorem (in group1) Group_ZF_3_4_T2:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  shows
  IsAmonoid(AH//R,ProjFun2(AH,R,Op2))
  R{id(G)} = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
proof -
  have group0(AH,Op1) using Group_ZF_3_2_L10A group0_def
    by simp
  with A1 groupAssum isAbelian show
    IsAmonoid(AH//R,ProjFun2(AH,R,Op2))
    R{id(G)} = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
    using Group_ZF_3_3_L2 group0.Group_ZF_2_4_L3 Group_ZF_3_4_L13A
      Group_ZF_3_4_L16 monoid0.Group_ZF_2_2_T1 Group_ZF_2_2_L1
    by auto
qed
```

38.5 Shifting almost homomorphisms

In this this section we consider what happens if we multiply an almost homomorphism by a group element. We show that the resulting function is also an a. h., and almost equal to the original one. This is used only for slopes (integer a.h.) in Int_ZF_2 where we need to correct a positive slopes by adding a constant, so that it is at least 2 on positive integers.

If s is an almost homomorphism and c is some constant from the group, then $s \cdot c$ is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_5_L1:
  assumes A1: s ∈ AH and A2: c∈G and
  A3: r = {⟨x,s(x)·c⟩. x∈G}
  shows
  ∀x∈G. r(x) = s(x)·c
  r ∈ AH
  s ≅ r
proof -
  from A1 A2 A3 have I: r:G→G
    using AlmostHoms_def apply_funtype group_op_closed
    ZF_fun_from_total by auto
  with A3 show II: ∀x∈G. r(x) = s(x)·c
    using ZF_fun_from_tot_val by simp
  with isAbelian A1 A2 have III:
    ∀p ∈ G×G. δ(r,p) = δ(s,p)·c⁻¹
    using group_op_closed AlmostHoms_def apply_funtype
    HomDiff_def group0_4_L7 by auto
  have {δ(r,p). p ∈ G×G} ∈ Fin(G)
  proof -
    from A1 A2 have
      {δ(s,p). p ∈ G×G} ∈ Fin(G)   c⁻¹∈G
```

```
      using AlmostHoms_def inverse_in_group by auto
    then have {δ(s,p)·c⁻¹. p ∈ G×G} ∈ Fin(G)
      using group_oper_fun Finite1_L16AA
      by simp
    moreover from III have
      {δ(r,p). p ∈ G×G} = {δ(s,p)·c⁻¹. p ∈ G×G}
      by (rule ZF1_1_L4B)
    ultimately show thesis by simp
  qed
  with I show IV: r ∈ AH using AlmostHoms_def
    by simp
  from isAbelian A1 A2 I II have
    ∀n ∈ G. s(n)·(r(n))⁻¹ = c⁻¹
    using AlmostHoms_def apply_funtype group0_4_L6AB
    by auto
  then have {s(n)·(r(n))⁻¹. n∈G} = {c⁻¹. n∈G}
    by (rule ZF1_1_L4B)
  with A1 A2 IV show s ≅ r
    using group0_2_L1 monoid0.group0_1_L3A
      inverse_in_group Group_ZF_3_4_L12 by simp
qed

end
```

39 Direct product

```
theory DirectProduct_ZF imports func_ZF

begin
```

This theory considers the direct product of binary operations. Contributed by Seo Sanghyeon.

39.1 Definition

In group theory the notion of direct product provides a natural way of creating a new group from two given groups.

Given $(G, \cdot)$ and $(H, \circ)$ a new operation $(G \times H, \times)$ is defined as $(g, h) \times (g', h') = (g \cdot g', h \circ h')$.

```
definition
  DirectProduct(P,Q,G,H) ≡
  {⟨x,⟨P⟨fst(fst(x)),fst(snd(x))⟩ , Q⟨snd(fst(x)),snd(snd(x))⟩⟩⟩.
  x ∈ (G×H)×(G×H)}
```

We define a context called `direct0` which holds an assumption that P, Q are binary operations on G, H, resp. and denotes R as the direct product of (G, P) and (H, Q).

```
locale direct0 =
  fixes P Q G H
  assumes Pfun: P : G×G→G
  assumes Qfun: Q : H×H→H
  fixes R
  defines Rdef [simp]: R ≡ DirectProduct(P,Q,G,H)
```

The direct product of binary operations is a binary operation.

```
lemma (in direct0) DirectProduct_ZF_1_L1:
  shows R : (G×H)×(G×H)→G×H
proof -
  from Pfun Qfun have ∀x∈(G×H)×(G×H).
    ⟨P⟨fst(fst(x)),fst(snd(x))⟩,Q⟨snd(fst(x)),snd(snd(x))⟩⟩ ∈ G×H
    by auto
  then show thesis using ZF_fun_from_total DirectProduct_def
    by simp
qed
```

And it has the intended value.

```
lemma (in direct0) DirectProduct_ZF_1_L2:
  shows ∀x∈(G×H). ∀y∈(G×H).
  R⟨x,y⟩ = ⟨P⟨fst(x),fst(y)⟩,Q⟨snd(x),snd(y)⟩⟩
  using DirectProduct_def DirectProduct_ZF_1_L1 ZF_fun_from_tot_val
  by simp
```

And the value belongs to the set the operation is defined on.

```
lemma (in direct0) DirectProduct_ZF_1_L3:
  shows ∀x∈(G×H). ∀y∈(G×H). R⟨x,y⟩ ∈ G×H
  using DirectProduct_ZF_1_L1 by simp
```

39.2 Associative and commutative operations

If P and Q are both associative or commutative operations, the direct product of P and Q has the same property.

Direct product of commutative operations is commutative.

```
lemma (in direct0) DirectProduct_ZF_2_L1:
  assumes P {is commutative on} G and Q {is commutative on} H
  shows R {is commutative on} G×H
proof -
  from assms have ∀x∈(G×H). ∀y∈(G×H). R⟨x,y⟩ = R⟨y,x⟩
    using DirectProduct_ZF_1_L2 IsCommutative_def by simp
  then show thesis using IsCommutative_def by simp
qed
```

Direct product of associative operations is associative.

```
lemma (in direct0) DirectProduct_ZF_2_L2:
  assumes P {is associative on} G and Q {is associative on} H
```

```
  shows R {is associative on} G×H
proof -
  have ∀x∈G×H. ∀y∈G×H. ∀z∈G×H. R⟨R⟨x,y⟩,z⟩ =
    ⟨P⟨P⟨fst(x),fst(y)⟩,fst(z)⟩,Q⟨Q⟨snd(x),snd(y)⟩,snd(z)⟩⟩
    using DirectProduct_ZF_1_L2 DirectProduct_ZF_1_L3
    by auto
  moreover have ∀x∈G×H. ∀y∈G×H. ∀z∈G×H. R⟨x,R⟨y,z⟩⟩ =
    ⟨P⟨fst(x),P⟨fst(y),fst(z)⟩⟩,Q⟨snd(x),Q⟨snd(y),snd(z)⟩⟩⟩
    using DirectProduct_ZF_1_L2 DirectProduct_ZF_1_L3 by auto
  ultimately have ∀x∈G×H. ∀y∈G×H. ∀z∈G×H. R⟨R⟨x,y⟩,z⟩ = R⟨x,R⟨y,z⟩⟩
    using assms IsAssociative_def by simp
  then show thesis
    using DirectProduct_ZF_1_L1 IsAssociative_def by simp
qed

end
```

40 Ordered groups - introduction

```
theory OrderedGroup_ZF imports Group_ZF_1 AbelianGroup_ZF Finite_ZF_1
OrderedLoop_ZF

begin
```

This theory file defines and shows the basic properties of (partially or linearly) ordered groups. We show that in linearly ordered groups finite sets are bounded and provide a sufficient condition for bounded sets to be finite. This allows to show in `Int_ZF_IML.thy` that subsets of integers are bounded iff they are finite. Some theorems proven here are properties of ordered loops rather that groups. However, for now the development is independent from the material in the `OrderedLoop_ZF` theory, we just import the definitions of `NonnegativeSet` and `PositiveSet` from there.

40.1 Ordered groups

This section defines ordered groups and various related notions.

An ordered group is a group equipped with a partial order that is "translation invariant", that is if $a \leq b$ then $a \cdot g \leq b \cdot g$ and $g \cdot a \leq g \cdot b$.

```
definition
  IsAnOrdGroup(G,P,r) ≡
  (IsAgroup(G,P) ∧ r⊆G×G ∧ IsPartOrder(G,r) ∧ (∀g∈G. ∀a b.
  ⟨a,b⟩ ∈ r ⟶ ⟨P⟨ a,g⟩,P⟨ b,g⟩ ⟩ ∈ r ∧ ⟨ P⟨ g,a⟩,P⟨ g,b⟩ ⟩ ∈ r ) )
```

We also define the absolute value as a ZF-function that is the identity on G^+ and the group inverse on the rest of the group.

definition

```
AbsoluteValue(G,P,r) ≡ id(Nonnegative(G,P,r)) ∪
restrict(GroupInv(G,P),G - Nonnegative(G,P,r))
```

The odd functions are defined as those having property $f(a^{-1}) = (f(a))^{-1}$. This looks a bit strange in the multiplicative notation, I have to admit. For linearly ordered groups a function f defined on the set of positive elements iniquely defines an odd function of the whole group. This function is called an odd extension of f

definition

```
OddExtension(G,P,r,f) ≡
(f ∪ {⟨a, GroupInv(G,P)(f(GroupInv(G,P)(a)))⟩.
a ∈ GroupInv(G,P)(PositiveSet(G,P,r))} ∪
{⟨TheNeutralElement(G,P),TheNeutralElement(G,P)⟩})
```

We will use a similar notation for ordered groups as for the generic groups. `G+` denotes the set of nonnegative elements (that satisfy $1 \leq a$) and `G+` is the set of (strictly) positive elements. `-A` is the set inverses of elements from A. I hope that using additive notation for this notion is not too shocking here. The symbol `f°` denotes the odd extension of f. For a function defined on G_+ this is the unique odd function on G that is equal to f on G_+.

locale group3 =

```
fixes G and P and r

assumes ordGroupAssum: IsAnOrdGroup(G,P,r)

fixes unit (1)
defines unit_def [simp]: 1 ≡ TheNeutralElement(G,P)

fixes groper (infixl · 70)
defines groper_def [simp]: a · b ≡ P⟨ a,b⟩

fixes inv (_⁻¹  [90] 91)
defines inv_def [simp]: x⁻¹ ≡ GroupInv(G,P)(x)

fixes lesseq (infix ≤ 68)
defines lesseq_def [simp]: a ≤ b ≡ ⟨a,b⟩ ∈ r

fixes sless (infix < 68)
defines sless_def [simp]: a < b ≡ a≤b ∧ a≠b

fixes nonnegative (G⁺)
defines nonnegative_def [simp]: G⁺ ≡ Nonnegative(G,P,r)

fixes positive (G₊)
defines positive_def [simp]: G₊ ≡ PositiveSet(G,P,r)

fixes setinv (- _ 72)
```

```
  defines setninv_def [simp]: -A ≡ GroupInv(G,P)(A)

  fixes abs (| _ |)
  defines abs_def [simp]: |a| ≡ AbsoluteValue(G,P,r)(a)

  fixes oddext (_ °)
  defines oddext_def [simp]: f° ≡ OddExtension(G,P,r,f)
```

In group3 context we can use the theorems proven in the group0 context.

```
lemma (in group3) OrderedGroup_ZF_1_L1: shows group0(G,P)
  using ordGroupAssum IsAnOrdGroup_def group0_def by simp
```

Ordered group (carrier) is not empty. This is a property of monoids, but it is good to have it handy in the group3 context.

```
lemma (in group3) OrderedGroup_ZF_1_L1A: shows G≠0
  using OrderedGroup_ZF_1_L1 group0.group0_2_L1 monoid0.group0_1_L3A
  by blast
```

The next lemma is just to see the definition of the nonnegative set in our notation.

```
lemma (in group3) OrderedGroup_ZF_1_L2:
  shows g∈G⁺ ⟷ 1≤g
  using ordGroupAssum IsAnOrdGroup_def Nonnegative_def
  by auto
```

The next lemma is just to see the definition of the positive set in our notation.

```
lemma (in group3) OrderedGroup_ZF_1_L2A:
  shows g∈G₊ ⟷ (1≤g ∧ g≠1)
  using ordGroupAssum IsAnOrdGroup_def PositiveSet_def
  by auto
```

For total order if g is not in G^+, then it has to be less or equal the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L2B:
  assumes A1: r {is total on} G and A2: a∈G\G⁺
  shows a≤1 a<1
proof -
  from A2 have a∈G   1 ∈ G  ¬(1≤a)
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2 OrderedGroup_ZF_1_L2

    by auto
  with A1 show a≤1 a<1 using IsTotal_def by auto
qed
```

The group order is reflexive.

```
lemma (in group3) OrderedGroup_ZF_1_L3: assumes g∈G
  shows g≤g
  using ordGroupAssum assms IsAnOrdGroup_def IsPartOrder_def refl_def
```

```
  by simp
```

1 is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L3A: shows 1∈G+
  using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L3
    OrderedGroup_ZF_1_L1 group0.group0_2_L2 by simp
```

In this context $a \leq b$ implies that both a and b belong to G.

```
lemma (in group3) OrderedGroup_ZF_1_L4:
  assumes a≤b shows a∈G b∈G
  using ordGroupAssum assms IsAnOrdGroup_def by auto
```

Similarly in this context $a \leq b$ implies that both a and b belong to G.

```
lemma (in group3) less_are_members:
  assumes a<b shows a∈G b∈G
  using ordGroupAssum assms IsAnOrdGroup_def by auto
```

It is good to have transitivity handy.

```
lemma (in group3) Group_order_transitive:
  assumes A1: a≤b  b≤c shows a≤c
proof -
  from ordGroupAssum have trans(r)
    using IsAnOrdGroup_def IsPartOrder_def
    by simp
  moreover from A1 have ⟨ a,b⟩ ∈ r ∧ ⟨ b,c⟩ ∈ r by simp
  ultimately have ⟨ a,c⟩ ∈ r by (rule Fol1_L3)
  thus thesis by simp
qed
```

The order in an ordered group is antisymmetric.

```
lemma (in group3) group_order_antisym:
  assumes A1: a≤b  b≤a shows a=b
proof -
  from ordGroupAssum A1 have
    antisym(r)  ⟨a,b⟩ ∈ r  ⟨b,a⟩ ∈ r
    using IsAnOrdGroup_def IsPartOrder_def by auto
  then show a=b by (rule Fol1_L4)
qed
```

Transitivity for the strict order: if $a < b$ and $b \leq c$, then $a < c$.

```
lemma (in group3) OrderedGroup_ZF_1_L4A:
  assumes A1: a<b  and A2: b≤c
  shows a<c
proof -
  from A1 A2 have a≤b  b≤c by auto
  then have a≤c by (rule Group_order_transitive)
  moreover from A1 A2 have a≠c using group_order_antisym by auto
  ultimately show a<c by simp
```

qed

If $a < b$ then it's not true that $b \leq a$.

```
lemma (in group3) ls_not_leq: assumes a<b shows ¬(b≤a)
proof -
  { assume b≤a
    with assms have a<a using OrderedGroup_ZF_1_L4A by blast
  } thus ¬(b≤a) by auto
qed
```

Another version of transitivity for the strict order: if $a \leq b$ and $b < c$, then $a < c$.

```
lemma (in group3) group_strict_ord_transit:
  assumes A1: a≤b and A2: b<c
  shows a<c
proof -
  from A1 A2 have a≤b  b≤c by auto
  then have  a≤c by (rule Group_order_transitive)
  moreover from A1 A2 have a≠c using group_order_antisym by auto
  ultimately show a<c by simp
qed
```

The order is translation invariant.

```
lemma (in group3) ord_transl_inv: assumes a≤b c∈G
  shows a·c ≤ b·c and c·a ≤ c·b
  using ordGroupAssum assms unfolding IsAnOrdGroup_def by auto
```

Strict order is preserved by translations.

```
lemma (in group3) group_strict_ord_transl_inv:
  assumes a<b and c∈G
  shows a·c < b·c and c·a < c·b
  using assms ord_transl_inv OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
group0.group0_2_L19
  by auto
```

If the group order is total, then the group is ordered linearly.

```
lemma (in group3) group_ord_total_is_lin:
  assumes r {is total on} G
  shows IsLinOrder(G,r)
  using assms ordGroupAssum IsAnOrdGroup_def Order_ZF_1_L3
  by simp
```

For linearly ordered groups elements in the nonnegative set are greater than those in the complement.

```
lemma (in group3) OrderedGroup_ZF_1_L4B:
  assumes r {is total on} G
  and a∈G⁺ and b ∈ G-G⁺
```

```
  shows b≤a
proof -
  from assms have b≤1 1≤a
    using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L2B by auto
  then show thesis by (rule Group_order_transitive)
qed
```

If $a \leq 1$ and $a \neq 1$, then $a \in G \setminus G^+$.

```
lemma (in group3) OrderedGroup_ZF_1_L4C:
  assumes A1: a≤1 and A2: a≠1
  shows a ∈ G\G+
proof -
  { assume a ∉ G-G+
    with ordGroupAssum A1 A2 have False
      using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L2
 OrderedGroup_ZF_1_L4 IsAnOrdGroup_def IsPartOrder_def antisym_def
      by auto
  } thus thesis by auto
qed
```

If a is smaller than the neutral element then a is in the complement of the set of nonnegative elements. This is just `OrderedGroup_ZF_1_L4C` with assumptions in a different form.

```
corollary (in group3) smaller_one_negative: assumes a<1
  shows a ∈ G\G+ using assms OrderedGroup_ZF_1_L4C by simp
```

An element smaller than an element in $G \setminus G^+$ is in $G \setminus G^+$.

```
lemma (in group3) OrderedGroup_ZF_1_L4D:
  assumes A1: a∈G\G+ and A2: b≤a
  shows b∈G\G+
proof -
  { assume b ∉ G\G+
    with A2 have 1≤b b≤a
      using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L2 by auto
    then have 1≤a by (rule Group_order_transitive)
    with A1 have False using OrderedGroup_ZF_1_L2 by simp
  } thus thesis by auto
qed
```

The nonnegative set is contained in the group.

```
lemma (in group3) OrderedGroup_ZF_1_L4E: shows G+ ⊆ G
  using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L4 by auto
```

The positive set is contained in the nonnegative set, hence in the group.

```
lemma (in group3) pos_set_in_gr: shows G+ ⊆ G+ and G+ ⊆ G
  using OrderedGroup_ZF_1_L2A OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L4E
  by auto
```

Taking the inverse on both sides reverses the inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L5:
  assumes A1: a≤b shows b⁻¹≤a⁻¹
proof -
  from A1 have T1: a∈G b∈G a⁻¹∈G b⁻¹∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  with A1 ordGroupAssum have a·a⁻¹≤b·a⁻¹ using IsAnOrdGroup_def
    by simp
  with T1 ordGroupAssum have b⁻¹·1≤b⁻¹·(b·a⁻¹)
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6 IsAnOrdGroup_def
    by simp
  with T1 show thesis using
    OrderedGroup_ZF_1_L1 group0.group0_2_L2 group0.group_oper_assoc
    group0.group0_2_L6 by simp
qed
```

Taking the inverse on both sides reverses the strict inequality.

```
lemma (in group3) inv_both_strict_ineq:
  assumes a<b shows b⁻¹<a⁻¹
proof -
  from assms have b⁻¹≤a⁻¹ using OrderedGroup_ZF_1_L5 by simp
  from assms have a∈G b∈G a≠b using less_are_members by simp_all
  with ‹b⁻¹≤a⁻¹› show b⁻¹<a⁻¹
    using OrderedGroup_ZF_1_L1 group0.el_neq_inv_neq by auto
qed
```

If an element is less or equal than the unit, then its inverse is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L5A:
  assumes A1: a≤1 shows 1≤a⁻¹
proof -
  from A1 have 1⁻¹≤a⁻¹ using OrderedGroup_ZF_1_L5
    by simp
  then show thesis using OrderedGroup_ZF_1_L1 group0.group_inv_of_one

    by simp
qed
```

If an the inverse of an element is greater that the unit, then the element is smaller.

```
lemma (in group3) OrderedGroup_ZF_1_L5AA:
  assumes A1: a∈G and A2: 1≤a⁻¹
  shows a≤1
proof -
  from A2 have (a⁻¹)⁻¹≤1⁻¹ using OrderedGroup_ZF_1_L5
    by simp
  with A1 show a≤1
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv group0.group_inv_of_one
```

```
    by simp
qed
```

If an element is nonnegative, then the inverse is not greater that the unit. Also shows that nonnegative elements cannot be negative

```
lemma (in group3) OrderedGroup_ZF_1_L5AB:
  assumes A1: 1≤a shows a⁻¹≤1 and ¬(a≤1 ∧ a≠1)
proof -
  from A1 have a⁻¹≤1⁻¹
    using OrderedGroup_ZF_1_L5 by simp
  then show a⁻¹≤1 using OrderedGroup_ZF_1_L1 group0.group_inv_of_one
    by simp
  { assume a≤1 and a≠1
    with A1 have False using group_order_antisym
      by blast
  } then show ¬(a≤1 ∧ a≠1) by auto
qed
```

If an element is positive, then its inverse is negative.

```
lemma (in group3) pos_inv_neg: assumes  1<a
  shows a⁻¹<1
proof -
  from assms have a∈G and a≠1
    using less_are_members(2) by auto
  with assms show a⁻¹<1
    using OrderedGroup_ZF_1_L1 group0.group0_2_L8B OrderedGroup_ZF_1_L5AB(1)
    by simp
qed
```

If two elements are greater or equal than the unit, then the inverse of one is not greater than the other.

```
lemma (in group3) OrderedGroup_ZF_1_L5AC:
  assumes A1: 1≤a  1≤b
  shows a⁻¹ ≤ b
proof -
  from A1 have a⁻¹≤1  1≤b
    using OrderedGroup_ZF_1_L5AB by auto
  then show a⁻¹ ≤ b by (rule Group_order_transitive)
qed
```

40.2 Inequalities

This section developes some simple tools to deal with inequalities.

Taking negative on both sides reverses the inequality, case with an inverse on one side.

```
lemma (in group3) OrderedGroup_ZF_1_L5AD:
  assumes A1: b ∈ G and A2: a≤b⁻¹
```

```
  shows b ≤ a⁻¹
proof -
  from A2 have (b⁻¹)⁻¹ ≤ a⁻¹
    using OrderedGroup_ZF_1_L5 by simp
  with A1 show b ≤ a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

We can cancel the same element on both sides of an inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L5AE:
  assumes A1: a∈G  b∈G  c∈G and A2: a·b ≤ a·c
  shows b≤c
proof -
  from ordGroupAssum A1 A2 have a⁻¹·(a·b) ≤ a⁻¹·(a·c)
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def

    by simp
  with A1 show b≤c
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by simp
qed
```

We can cancel the same element on both sides of an inequality, right side.

```
lemma (in group3) ineq_cancel_right:
  assumes a∈G  b∈G  c∈G and a·b ≤ c·b
  shows a≤c
proof -
  from assms(2,4) have (a·b)·b⁻¹ ≤ (c·b)·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group ord_transl_inv(1)
by simp
  with assms(1,2,3) show a≤c using OrderedGroup_ZF_1_L1 group0.inv_cancel_two(2)

    by auto
qed
```

We can cancel the same element on both sides of an inequality, a version with an inverse on both sides.

```
lemma (in group3) OrderedGroup_ZF_1_L5AF:
  assumes A1: a∈G  b∈G  c∈G and A2: a·b⁻¹ ≤ a·c⁻¹
  shows c≤b
proof -
  from A1 A2 have (c⁻¹)⁻¹ ≤ (b⁻¹)⁻¹
     using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      OrderedGroup_ZF_1_L5AE OrderedGroup_ZF_1_L5 by simp
  with A1 show c≤b
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv by simp
qed
```

Taking negative on both sides reverses the inequality, another case with an inverse on one side.

```
lemma (in group3) OrderedGroup_ZF_1_L5AG:
  assumes A1: a ∈ G and A2: a⁻¹≤b
  shows b⁻¹ ≤ a
proof -
  from A2 have b⁻¹ ≤ (a⁻¹)⁻¹
    using OrderedGroup_ZF_1_L5 by simp
  with A1 show b⁻¹ ≤ a
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

We can multiply the sides of two inequalities.

```
lemma (in group3) OrderedGroup_ZF_1_L5B:
  assumes A1: a≤b and A2: c≤d
  shows a·c ≤ b·d
proof -
  from A1 A2 have c∈G b∈G using OrderedGroup_ZF_1_L4 by auto
  with A1 A2 ordGroupAssum have a·c≤ b·c b·c≤b·d
    using IsAnOrdGroup_def by auto
  then show a·c ≤ b·d by (rule Group_order_transitive)
qed
```

The set of nonnegative elements is closed with respect to the group operation.

```
lemma (in group3) group_nonnegative_closed: assumes a∈G⁺ b∈G⁺
  shows a·b ∈ G⁺
proof -
  from assms have 1·1 ≤ a·b
    using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L5B by simp
  then show thesis using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    OrderedGroup_ZF_1_L2 by simp
qed
```

We can replace first of the factors on one side of an inequality with a greater one.

```
lemma (in group3) OrderedGroup_ZF_1_L5C:
  assumes A1: c∈G and A2: a≤b·c and A3: b≤b₁
  shows a≤b₁·c
proof -
  from A1 A3 have b·c ≤ b₁·c
    using OrderedGroup_ZF_1_L3 OrderedGroup_ZF_1_L5B by simp
  with A2 show a≤b₁·c by (rule Group_order_transitive)
qed
```

We can replace second of the factors on one side of an inequality with a greater one.

```
lemma (in group3) OrderedGroup_ZF_1_L5D:
  assumes A1: b∈G and A2: a ≤ b·c and A3: c≤b₁
  shows a ≤ b·b₁
proof -
  from A1 A3 have b·c ≤ b·b₁
    using OrderedGroup_ZF_1_L3 OrderedGroup_ZF_1_L5B by auto
  with A2 show a≤b·b₁ by (rule Group_order_transitive)
qed
```

We can replace factors on one side of an inequality with greater ones.

```
lemma (in group3) OrderedGroup_ZF_1_L5E:
  assumes A1: a ≤ b·c and A2: b≤b₁  c≤c₁
  shows a ≤ b₁·c₁
proof -
  from A2 have b·c ≤ b₁·c₁ using OrderedGroup_ZF_1_L5B
    by simp
  with A1 show a≤b₁·c₁ by (rule Group_order_transitive)
qed
```

We don't decrease an element of the group by multiplying by one that is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L5F:
  assumes A1: 1≤a and A2: b∈G
  shows b≤a·b  b≤b·a
proof -
  from ordGroupAssum A1 A2 have
    1·b≤a·b  b·1≤b·a
    using IsAnOrdGroup_def by auto
  with A2 show b≤a·b  b≤b·a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
qed
```

We can multiply the right hand side of an inequality by a nonnegative element.

```
lemma (in group3) OrderedGroup_ZF_1_L5G: assumes A1: a≤b
  and A2: 1≤c shows a≤b·c  a≤c·b
proof -
  from A1 A2 have I: b≤b·c  and II: b≤c·b
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L5F by auto
  from A1 I show a≤b·c by (rule Group_order_transitive)
  from A1 II show a≤c·b by (rule Group_order_transitive)
qed
```

We can put two elements on the other side of inequality, changing their sign.

```
lemma (in group3) OrderedGroup_ZF_1_L5H:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ≤ c
  shows
```

```
  a ≤ c·b
  c⁻¹·a ≤ b
proof -
  from A2 have T: c∈G  c⁻¹ ∈ G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  from ordGroupAssum A1 A2 have a·b⁻¹·b ≤ c·b
    using IsAnOrdGroup_def by simp
  with A1 show a ≤ c·b
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by simp
  with ordGroupAssum A2 T have c⁻¹·a ≤ c⁻¹·(c·b)
    using IsAnOrdGroup_def by simp
  with A1 T show c⁻¹·a ≤ b
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by simp
qed
```

We can multiply the sides of one inequality by inverse of another.

```
lemma (in group3) OrderedGroup_ZF_1_L5I:
  assumes a≤b and c≤d
  shows a·d⁻¹ ≤ b·c⁻¹
  using assms OrderedGroup_ZF_1_L5 OrderedGroup_ZF_1_L5B
  by simp
```

We can put an element on the other side of an inequality changing its sign, version with the inverse.

```
lemma (in group3) OrderedGroup_ZF_1_L5J:
  assumes A1: a∈G  b∈G and A2: c ≤ a·b⁻¹
  shows c·b ≤ a
proof -
  from ordGroupAssum A1 A2 have c·b ≤ a·b⁻¹·b
    using IsAnOrdGroup_def by simp
  with A1 show c·b ≤ a
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by simp
qed
```

We can put an element on the other side of an inequality changing its sign, version with the inverse.

```
lemma (in group3) OrderedGroup_ZF_1_L5JA:
  assumes A1: a∈G  b∈G and A2: c ≤ a⁻¹·b
  shows a·c≤ b
proof -
  from ordGroupAssum A1 A2 have a·c ≤ a·(a⁻¹·b)
    using IsAnOrdGroup_def by simp
  with A1 show a·c≤ b
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
```

```
    by simp
qed
```

A special case of `OrderedGroup_ZF_1_L5J` where $c = 1$.

```
corollary (in group3) OrderedGroup_ZF_1_L5K:
  assumes A1: a∈G  b∈G and A2: 1 ≤ a·b⁻¹
  shows b ≤ a
proof -
  from A1 A2 have 1·b ≤ a
    using OrderedGroup_ZF_1_L5J by simp
  with A1 show b ≤ a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

A special case of `OrderedGroup_ZF_1_L5JA` where $c = 1$.

```
corollary (in group3) OrderedGroup_ZF_1_L5KA:
  assumes A1: a∈G  b∈G and A2: 1 ≤ a⁻¹·b
  shows a ≤ b
proof -
  from A1 A2 have a·1 ≤ b
    using OrderedGroup_ZF_1_L5JA by simp
  with A1 show a ≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

If the order is total, the elements that do not belong to the positive set are negative. We also show here that the group inverse of an element that does not belong to the nonnegative set does belong to the nonnegative set.

```
lemma (in group3) OrderedGroup_ZF_1_L6:
  assumes A1: r {is total on} G and A2: a∈G\G⁺
  shows a≤1  a⁻¹ ∈ G⁺  restrict(GroupInv(G,P),G-G⁺)(a) ∈ G⁺
proof -
  from A2 have T1: a∈G a∉G⁺ 1∈G
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2 by auto
  with A1 show a≤1 using OrderedGroup_ZF_1_L2 IsTotal_def
    by auto
  then show a⁻¹ ∈ G⁺ using OrderedGroup_ZF_1_L5A OrderedGroup_ZF_1_L2
    by simp
  with A2 show restrict(GroupInv(G,P),G-G⁺)(a) ∈ G⁺
    using restrict by simp
qed
```

If a property is invariant with respect to taking the inverse and it is true on the nonnegative set, than it is true on the whole group.

```
lemma (in group3) OrderedGroup_ZF_1_L7:
  assumes A1: r {is total on} G
```

```
  and A2: ∀a∈G⁺.∀b∈G⁺. Q(a,b)
  and A3: ∀a∈G.∀b∈G. Q(a,b)⟶Q(a⁻¹,b)
  and A4: ∀a∈G.∀b∈G. Q(a,b)⟶Q(a,b⁻¹)
  and A5: a∈G b∈G
  shows Q(a,b)
proof -
  { assume A6: a∈G⁺ have Q(a,b)
    proof -
      { assume b∈G⁺
 with A6 A2 have Q(a,b) by simp }
      moreover
      { assume b∉G⁺
 with A1 A2 A4 A5 A6 have Q(a,(b⁻¹)⁻¹)
   using OrderedGroup_ZF_1_L6 OrderedGroup_ZF_1_L1 group0.inverse_in_group
   by simp
 with A5 have Q(a,b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
   by simp }
      ultimately show Q(a,b) by auto
    qed }
  moreover
  { assume a∉G⁺
    with A1 A5 have T1: a⁻¹ ∈ G⁺ using OrderedGroup_ZF_1_L6 by simp
    have Q(a,b)
    proof -
      { assume b∈G⁺
 with A2 A3 A5 T1 have Q((a⁻¹)⁻¹,b)
   using OrderedGroup_ZF_1_L1 group0.inverse_in_group by simp
 with A5 have Q(a,b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
   by simp }
      moreover
      { assume b∉G⁺
 with A1 A2 A3 A4 A5 T1 have Q((a⁻¹)⁻¹,(b⁻¹)⁻¹)
   using OrderedGroup_ZF_1_L6 OrderedGroup_ZF_1_L1 group0.inverse_in_group
   by simp
 with A5 have Q(a,b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
   by simp }
      ultimately show Q(a,b) by auto
    qed }
  ultimately show Q(a,b) by auto
qed
```

If a,b are an elements of an ordered group where the order is total, then $a\leq b$ or $b<a$.

```
lemma (in group3) OrdGroup_2cases: assumes r {is total on} G a∈G  b∈G
  shows a≤b ∨ b<a
  using assms IsTotal_def by auto
```

If a,b are an elements of an ordered group where the order is total, then $a<b$ or $a=b$ or $b\leq a$.

```
lemma (in group3) OrdGroup_3cases: assumes r {is total on} G a∈G  b∈G
  shows a<b ∨ a=b ∨ b<a
  using assms assms IsTotal_def by auto
```

A lemma about splitting the ordered group "plane" into 6 subsets. Useful for proofs by cases.

```
lemma (in group3) OrdGroup_6cases: assumes A1: r {is total on} G
  and A2:  a∈G  b∈G
  shows
  1≤a ∧ 1≤b  ∨  a≤1 ∧ b≤1  ∨
  a≤1 ∧ 1≤b ∧ 1 ≤ a·b  ∨ a≤1 ∧ 1≤b ∧ a·b ≤ 1  ∨
  1≤a ∧ b≤1 ∧ 1 ≤ a·b  ∨  1≤a ∧ b≤1 ∧ a·b ≤ 1
proof -
  from A1 A2 have
    1≤a ∨ a≤1
    1≤b ∨ b≤1
    1 ≤ a·b ∨ a·b ≤ 1
    using OrderedGroup_ZF_1_L1 group0.group_op_closed group0.group0_2_L2
      IsTotal_def by auto
  then show thesis by auto
qed
```

The next lemma shows what happens when one element of a totally ordered group is not greater or equal than another.

```
lemma (in group3) OrderedGroup_ZF_1_L8:
  assumes A1: r {is total on} G
  and A2: a∈G  b∈G
  and A3: ¬(a≤b)
  shows b ≤ a  a⁻¹ ≤ b⁻¹  a≠b  b<a

proof -
  from A1 A2 A3 show I: b ≤ a using IsTotal_def
    by auto
  then show a⁻¹ ≤ b⁻¹ using OrderedGroup_ZF_1_L5 by simp
  from A2 have a ≤ a using OrderedGroup_ZF_1_L3 by simp
  with I A3 show a≠b  b < a by auto
qed
```

If one element is greater or equal and not equal to another, then it is not smaller or equal.

```
lemma (in group3) OrderedGroup_ZF_1_L8AA:
  assumes A1: a≤b and A2: a≠b
  shows ¬(b≤a)
proof -
  { note A1
    moreover assume b≤a
    ultimately have a=b by (rule group_order_antisym)
    with A2 have False by simp
```

```
  } thus ¬(b≤a) by auto
qed
```

A special case of `OrderedGroup_ZF_1_L8` when one of the elements is the unit.

```
corollary (in group3) OrderedGroup_ZF_1_L8A:
  assumes A1: r {is total on} G
  and A2: a∈G and A3: ¬(1≤a)
  shows 1 ≤ a⁻¹  1≠a  a≤1
proof -
  from A1 A2 A3 have I:
    r {is total on} G
    1∈G  a∈G
     ¬(1≤a)
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  then have 1⁻¹ ≤ a⁻¹
    by (rule OrderedGroup_ZF_1_L8)
  then show 1 ≤ a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_one by simp
  from I show 1≠a by (rule OrderedGroup_ZF_1_L8)
  from A1 I show a≤1 using IsTotal_def
    by auto
qed
```

A negative element can not be nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L8B:
  assumes A1: a≤1  and A2: a≠1 shows ¬(1≤a)
proof -
  { assume  1≤a
    with A1 have a=1 using group_order_antisym
      by auto
    with A2 have False by simp
  } thus thesis by auto
qed
```

An element is greater or equal than another iff the difference is nonpositive.

```
lemma (in group3) OrderedGroup_ZF_1_L9:
  assumes A1: a∈G  b∈G
  shows a≤b ⟷ a·b⁻¹ ≤ 1
proof
  assume a ≤ b
  with ordGroupAssum A1 have a·b⁻¹ ≤ b·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group
    IsAnOrdGroup_def by simp
  with A1 show a·b⁻¹ ≤ 1
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6
    by simp
next assume A2: a·b⁻¹ ≤ 1
  with ordGroupAssum A1 have a·b⁻¹·b ≤ 1·b
```

```
    using IsAnOrdGroup_def by simp
  with A1 show a ≤ b
    using OrderedGroup_ZF_1_L1
      group0.inv_cancel_two group0.group0_2_L2
    by simp
qed
```

We can move an element to the other side of an inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L9A:
  assumes A1: a∈G  b∈G  c∈G
  shows a·b ≤ c  ⟷ a ≤ c·b⁻¹
proof
  assume a·b ≤ c
  with ordGroupAssum A1 have a·b·b⁻¹ ≤ c·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def
    by simp
  with A1 show a ≤ c·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two by simp
next assume a ≤ c·b⁻¹
  with ordGroupAssum A1 have a·b ≤ c·b⁻¹·b
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def
    by simp
  with A1 show a·b ≤ c
     using OrderedGroup_ZF_1_L1 group0.inv_cancel_two by simp
qed
```

A one side version of the previous lemma with weaker assuptions.

```
lemma (in group3) OrderedGroup_ZF_1_L9B:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ≤ c
  shows a ≤ c·b
proof -
  from A1 A2 have a∈G  b⁻¹∈G  c∈G
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      OrderedGroup_ZF_1_L4 by auto
  with A1 A2 show a ≤ c·b
    using OrderedGroup_ZF_1_L9A OrderedGroup_ZF_1_L1
      group0.group_inv_of_inv by simp
qed
```

We can put en element on the other side of inequality, changing its sign.

```
lemma (in group3) OrderedGroup_ZF_1_L9C:
  assumes A1: a∈G  b∈G and A2: c≤a·b
  shows
  c·b⁻¹ ≤ a
  a⁻¹·c ≤ b
proof -
  from ordGroupAssum A1 A2 have
    c·b⁻¹ ≤ a·b·b⁻¹
    a⁻¹·c ≤ a⁻¹·(a·b)
```

```
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def
    by auto
  with A1 show
    c·b⁻¹ ≤ a
    a⁻¹·c ≤ b
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by auto
qed
```

If an element is greater or equal than another then the difference is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L9D: assumes A1: a≤b
  shows 1 ≤ b·a⁻¹
proof -
  from A1 have T: a∈G  b∈G   a⁻¹ ∈ G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  with ordGroupAssum A1 have a·a⁻¹ ≤ b·a⁻¹
    using IsAnOrdGroup_def by simp
  with T show 1 ≤ b·a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6
    by simp
qed
```

If an element is greater than another then the difference is positive.

```
lemma (in group3) OrderedGroup_ZF_1_L9E:
  assumes A1: a≤b  a≠b
  shows 1 ≤ b·a⁻¹  1 ≠ b·a⁻¹  b·a⁻¹ ∈ G₊
proof -
  from A1 have T: a∈G  b∈G using OrderedGroup_ZF_1_L4
    by auto
  from A1 show I: 1 ≤ b·a⁻¹ using OrderedGroup_ZF_1_L9D
    by simp
  { assume b·a⁻¹ = 1
    with T have a=b
      using OrderedGroup_ZF_1_L1 group0.group0_2_L11A
      by auto
    with A1 have False by simp
  } then show 1 ≠ b·a⁻¹ by auto
  then have b·a⁻¹ ≠ 1 by auto
  with I show b·a⁻¹ ∈ G₊ using OrderedGroup_ZF_1_L2A
    by simp
qed
```

If the difference is nonnegative, then $a \leq b$.

```
lemma (in group3) OrderedGroup_ZF_1_L9F:
  assumes A1: a∈G  b∈G and A2: 1 ≤ b·a⁻¹
  shows a≤b
proof -
```

```
  from A1 A2 have 1·a ≤ b
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L9A
    by simp
  with A1 show a≤b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

If we increase the middle term in a product, the whole product increases.

```
lemma (in group3) OrderedGroup_ZF_1_L10:
  assumes a∈G  b∈G and c≤d
  shows a·c·b ≤ a·d·b
  using ordGroupAssum assms IsAnOrdGroup_def by simp
```

A product of (strictly) positive elements is not the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L11:
  assumes A1: 1≤a  1≤b
  and A2: 1 ≠ a  1 ≠ b
  shows 1 ≠ a·b
proof -
  from A1 have T1: a∈G  b∈G
    using OrderedGroup_ZF_1_L4 by auto
  { assume 1 = a·b
    with A1 T1 have a≤1  1≤a
      using OrderedGroup_ZF_1_L1 group0.group0_2_L9 OrderedGroup_ZF_1_L5AA

      by auto
    then have a = 1 by (rule group_order_antisym)
    with A2 have False by simp
  } then show 1 ≠ a·b by auto
qed
```

A product of nonnegative elements is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L12:
  assumes A1: 1 ≤ a  1 ≤ b
  shows 1 ≤ a·b
proof -
  from A1 have 1·1 ≤ a·b
    using  OrderedGroup_ZF_1_L5B by simp
  then show 1 ≤ a·b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

If a is not greater than b, then 1 is not greater than $b \cdot a^{-1}$.

```
lemma (in group3) OrderedGroup_ZF_1_L12A:
  assumes A1: a≤b shows 1 ≤ b·a⁻¹
proof -
```

```
  from A1 have T: 1 ∈ G  a∈G  b∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  with A1 have 1·a ≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
  with T show 1 ≤ b·a⁻¹ using OrderedGroup_ZF_1_L9A
    by simp
qed
```

We can move an element to the other side of a strict inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L12B:
  assumes A1: a∈G  b∈G and  A2: a·b⁻¹ < c
  shows a < c·b
proof -
  from A1 A2 have a·b⁻¹·b < c·b
    using group_strict_ord_transl_inv by auto
  moreover from A1 have a·b⁻¹·b = a
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by simp
  ultimately show a < c·b
    by auto
qed
```

We can multiply the sides of two inequalities, first of them strict and we get a strict inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L12C:
  assumes A1: a<b and A2: c≤d
  shows a·c < b·d
proof -
  from A1 A2 have T: a∈G  b∈G  c∈G  d∈G
    using OrderedGroup_ZF_1_L4 by auto
  with ordGroupAssum A2 have a·c ≤ a·d
    using IsAnOrdGroup_def by simp
  moreover from A1 T have a·d < b·d
    using group_strict_ord_transl_inv by simp
  ultimately show a·c < b·d
    by (rule group_strict_ord_transit)
qed
```

We can multiply the sides of two inequalities, second of them strict and we get a strict inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L12D:
  assumes A1: a≤b and A2: c<d
  shows a·c < b·d
proof -
  from A1 A2 have T: a∈G  b∈G  c∈G  d∈G
    using OrderedGroup_ZF_1_L4 by auto
```

```
  with A2 have a·c < a·d
    using group_strict_ord_transl_inv by simp
  moreover from ordGroupAssum A1 T have a·d ≤ b·d
     using IsAnOrdGroup_def by simp
  ultimately show a·c < b·d
    by (rule OrderedGroup_ZF_1_L4A)
qed
```

If two elements of the group are smaller than the neutral element then their product is also smaller.

```
lemma (in group3) group_less_less: assumes a<1 b<1
  shows a·b < 1
proof -
  from assms have a·b < 1·1 using OrderedGroup_ZF_1_L12D
    by simp
  then show a·b < 1 using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

If the order is total the complement of the set of nonnegative elements is closed with respect to the group operation.

```
lemma (in group3) group_negative_closed:
  assumes r {is total on} G a∈G\G⁺ b∈G\G⁺
  shows a·b ∈ G\G⁺
proof -
  from assms have a·b < 1·1
    using OrderedGroup_ZF_1_L2B(2) OrderedGroup_ZF_1_L12D by simp
  then show a·b ∈ G\G⁺ using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    smaller_one_negative by simp
qed
```

40.3 The set of positive elements

In this section we study G_+ - the set of elements that are (strictly) greater than the unit. The most important result is that every linearly ordered group can decomposed into $\{1\}$, G_+ and the set of those elements $a \in G$ such that $a^{-1} \in G_+$. Another property of linearly ordered groups that we prove here is that if $G_+ \neq \emptyset$, then it is infinite. This allows to show that nontrivial linearly ordered groups are infinite.

The positive set is closed under the group operation.

```
lemma (in group3) OrderedGroup_ZF_1_L13: shows G₊ {is closed under}
P
proof -
  { fix a b assume a∈G₊  b∈G₊
    then have T1: 1 ≤ a·b   and 1 ≠ a·b
      using PositiveSet_def OrderedGroup_ZF_1_L11 OrderedGroup_ZF_1_L12
      by auto
```

```
    moreover from T1 have a·b ∈ G
      using OrderedGroup_ZF_1_L4 by simp
    ultimately have a·b ∈ G₊ using PositiveSet_def by simp
  } then show G₊ {is closed under} P using IsOpClosed_def
    by simp
qed
```

For totally ordered groups every nonunit element is positive or its inverse is positive.

```
lemma (in group3) OrderedGroup_ZF_1_L14:
  assumes A1: r {is total on} G and A2: a∈G
  shows a=1 ∨ a∈G₊ ∨ a⁻¹∈G₊
proof -
  { assume A3: a≠1
    moreover from A1 A2 have a≤1 ∨ 1≤a
      using IsTotal_def OrderedGroup_ZF_1_L1 group0.group0_2_L2
      by simp
    moreover from A3 A2 have T1: a⁻¹ ≠ 1
      using OrderedGroup_ZF_1_L1 group0.group0_2_L8B
      by simp
    ultimately have a⁻¹∈G₊ ∨ a∈G₊
      using OrderedGroup_ZF_1_L5A OrderedGroup_ZF_1_L2A
      by auto
  } thus a=1 ∨ a∈G₊ ∨ a⁻¹∈G₊ by auto
qed
```

If an element belongs to the positive set, then it is not the unit and its inverse does not belong to the positive set.

```
lemma (in group3) OrderedGroup_ZF_1_L15:
   assumes A1: a∈G₊  shows a≠1  a⁻¹∉G₊
proof -
  from A1 show T1: a≠1 using PositiveSet_def by auto
  { assume a⁻¹ ∈ G₊
    with A1 have a≤1  1≤a
      using OrderedGroup_ZF_1_L5AA PositiveSet_def by auto
    then have a=1 by (rule group_order_antisym)
    with T1 have False by simp
  } then show a⁻¹∉G₊ by auto
qed
```

If a^{-1} is positive, then a can not be positive or the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L16:
  assumes A1: a∈G and A2: a⁻¹∈G₊ shows a≠1  a∉G₊
proof -
  from A2 have a⁻¹≠1  (a⁻¹)⁻¹ ∉ G₊
    using OrderedGroup_ZF_1_L15 by auto
  with A1 show a≠1  a∉G₊
    using OrderedGroup_ZF_1_L1 group0.group0_2_L8C group0.group_inv_of_inv
```

```
    by auto
qed
```

For linearly ordered groups each element is either the unit, positive or its inverse is positive.

```
lemma (in group3) OrdGroup_decomp:
  assumes A1: r {is total on} G and A2: a∈G
  shows Exactly_1_of_3_holds (a=1,a∈G+,a⁻¹∈G+)
proof -
  from A1 A2 have a=1 ∨ a∈G+ ∨ a⁻¹∈G+
    using OrderedGroup_ZF_1_L14 by simp
  moreover from A2 have a=1 ⟶ (a∉G+ ∧ a⁻¹∉G+)
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_one
    PositiveSet_def by simp
  moreover from A2 have a∈G+ ⟶ (a≠1 ∧ a⁻¹∉G+)
    using OrderedGroup_ZF_1_L15 by simp
  moreover from A2 have a⁻¹∈G+ ⟶ (a≠1 ∧ a∉G+)
    using OrderedGroup_ZF_1_L16 by simp
  ultimately show Exactly_1_of_3_holds (a=1,a∈G+,a⁻¹∈G+)
    by (rule Fol1_L5)
qed
```

A if a is a nonunit element that is not positive, then a^{-1} is is positive. This is useful for some proofs by cases.

```
lemma (in group3) OrdGroup_cases:
  assumes A1: r {is total on} G and A2: a∈G
  and A3: a≠1  a∉G+
  shows a⁻¹ ∈ G+
proof -
  from A1 A2 have a=1 ∨ a∈G+ ∨ a⁻¹∈G+
    using OrderedGroup_ZF_1_L14 by simp
  with A3 show a⁻¹ ∈ G+ by auto
qed
```

Elements from $G \setminus G_+$ are not greater that the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L17:
  assumes A1: r {is total on} G and A2: a ∈ G-G+
  shows a≤1
proof -
  { assume a=1
    with A2 have a≤1 using OrderedGroup_ZF_1_L3 by simp }
  moreover
  { assume a≠1
    with A1 A2 have a≤1
      using PositiveSet_def OrderedGroup_ZF_1_L8A
      by auto }
  ultimately show a≤1 by auto
qed
```

The next lemma allows to split proofs that something holds for all $a \in G$ into cases $a = 1$, $a \in G_+$, $-a \in G_+$.

```
lemma (in group3) OrderedGroup_ZF_1_L18:
  assumes A1: r {is total on} G and A2: b∈G
  and A3: Q(1) and A4: ∀a∈G+. Q(a) and A5: ∀a∈G+. Q(a⁻¹)
  shows Q(b)
proof -
  from A1 A2 A3 A4 A5 have Q(b) ∨ Q((b⁻¹)⁻¹)
    using OrderedGroup_ZF_1_L14 by auto
  with A2 show Q(b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

All elements greater or equal than an element of G_+ belong to G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L19:
  assumes A1: a ∈ G+ and A2: a≤b
  shows b ∈ G+
proof -
  from A1 have I: 1≤a  and II: a≠1
    using OrderedGroup_ZF_1_L2A by auto
  from I A2 have 1≤b by (rule Group_order_transitive)
  moreover have b≠1
  proof -
    { assume b=1
      with I A2 have 1≤a  a≤1
 by auto
      then have 1=a by (rule group_order_antisym)
      with II have False by simp
    } then show b≠1 by auto
  qed
  ultimately show b ∈ G+
    using OrderedGroup_ZF_1_L2A by simp
qed
```

The inverse of an element of G_+ cannot be in G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L20:
  assumes A1: r {is total on} G and A2: a ∈ G+
  shows a⁻¹ ∉ G+
proof -
  from A2 have a∈G using PositiveSet_def
    by simp
  with A1 have Exactly_1_of_3_holds (a=1,a∈G+,a⁻¹∈G+)
    using OrdGroup_decomp by simp
  with A2 show a⁻¹ ∉ G+ by (rule Fol1_L7)
qed
```

The set of positive elements of a nontrivial linearly ordered group is not empty.

```
lemma (in group3) OrderedGroup_ZF_1_L21:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  shows G₊ ≠ 0
proof -
  have 1 ∈ G using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
  with A2 obtain a where a∈G  a≠1 by auto
  with A1 have a∈G₊ ∨ a⁻¹∈G₊
    using OrderedGroup_ZF_1_L14 by auto
  then show G₊ ≠ 0 by auto
qed
```

If b $\in$G$_+$, then $a < a \cdot b$. Multiplying a by a positive elemnt increases a.

```
lemma (in group3) OrderedGroup_ZF_1_L22:
  assumes A1: a∈G  b∈G₊
  shows a≤a·b  a ≠ a·b  a·b ∈ G
proof -
  from ordGroupAssum A1 have a·1 ≤ a·b
    using OrderedGroup_ZF_1_L2A IsAnOrdGroup_def
    by simp
  with A1 show a≤a·b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
  then show a·b ∈ G
    using OrderedGroup_ZF_1_L4 by simp
  { from A1 have a∈G  b∈G
      using PositiveSet_def by auto
    moreover assume a = a·b
    ultimately have b = 1
      using OrderedGroup_ZF_1_L1 group0.group0_2_L7
      by simp
    with A1 have False using PositiveSet_def
      by simp
  } then show a ≠ a·b by auto
qed
```

If G is a nontrivial linearly ordered hroup, then for every element of G we can find one in G$_+$ that is greater or equal.

```
lemma (in group3) OrderedGroup_ZF_1_L23:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  and A3: a∈G
  shows ∃b∈G₊. a≤b
proof -
  { assume A4: a∈G₊ then have a≤a
      using PositiveSet_def OrderedGroup_ZF_1_L3
      by simp
    with A4 have ∃b∈G₊. a≤b by auto }
  moreover
  { assume a∉G₊
```

```
    with A1 A3 have I: a≤1 using OrderedGroup_ZF_1_L17
      by simp
    from A1 A2 obtain b where II: b∈G+
      using OrderedGroup_ZF_1_L21 by auto
    then have 1≤b using PositiveSet_def by simp
    with I have a≤b by (rule Group_order_transitive)
    with II have ∃b∈G+. a≤b by auto }
  ultimately show thesis by auto
qed
```

The G^+ is G_+ plus the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L24: shows G+ = G+∪{1}
  using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L2A OrderedGroup_ZF_1_L3A
  by auto
```

What is $-G_+$, really?

```
lemma (in group3) OrderedGroup_ZF_1_L25: shows
  (-G+) = {a⁻¹. a∈G+}
  (-G+) ⊆ G
proof -
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  moreover have G+ ⊆ G using PositiveSet_def by auto
  ultimately show
    (-G+) = {a⁻¹. a∈G+}
    (-G+) ⊆ G
    using func_imagedef func1_1_L6 by auto
qed
```

If the inverse of a is in G_+, then a is in the inverse of G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L26:
  assumes A1: a∈G and A2: a⁻¹ ∈ G+
  shows a ∈ (-G+)
proof -
  from A1 have a⁻¹ ∈ G  a = (a⁻¹)⁻¹ using OrderedGroup_ZF_1_L1
    group0.inverse_in_group group0.group_inv_of_inv
    by auto
  with A2 show a ∈ (-G+) using OrderedGroup_ZF_1_L25
    by auto
qed
```

If a is in the inverse of G_+, then its inverse is in G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L27:
  assumes a ∈ (-G+)
  shows a⁻¹ ∈ G+
  using assms OrderedGroup_ZF_1_L25 PositiveSet_def
    OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
  by auto
```

A linearly ordered group can be decomposed into G_+, $\{1\}$ and $-G_+$

```
lemma (in group3) OrdGroup_decomp2:
  assumes A1: r {is total on} G
  shows
  G = G₊ ∪ (-G₊)∪ {1}
  G₊∩(-G₊) = 0
  1 ∉ G₊∪(-G₊)
proof -
  { fix a assume A2: a∈G
    with A1 have a∈G₊ ∨ a⁻¹∈G₊ ∨ a=1
      using OrderedGroup_ZF_1_L14 by auto
    with A2 have a∈G₊ ∨ a∈(-G₊) ∨ a=1
      using OrderedGroup_ZF_1_L26 by auto
    then have a ∈ (G₊ ∪ (-G₊)∪ {1})
      by auto
  } then have G ⊆ G₊ ∪ (-G₊)∪ {1}
    by auto
  moreover have G₊ ∪ (-G₊)∪ {1} ⊆ G
    using OrderedGroup_ZF_1_L25 PositiveSet_def
      OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  ultimately show G = G₊ ∪ (-G₊)∪ {1} by auto
  { let A = G₊∩(-G₊)
    assume G₊∩(-G₊) ≠ 0
    then have A≠0 by simp
    then obtain a where a∈A by blast
    then have False using OrderedGroup_ZF_1_L15 OrderedGroup_ZF_1_L27
      by auto
  } then show G₊∩(-G₊) = 0 by auto
  show 1 ∉ G₊∪(-G₊)
    using OrderedGroup_ZF_1_L27
      OrderedGroup_ZF_1_L1 group0.group_inv_of_one
      OrderedGroup_ZF_1_L2A by auto
qed
```

If $a \cdot b^{-1}$ is nonnegative, then $b \leq a$. This maybe used to recover the order from the set of nonnegative elements and serve as a way to define order by prescibing that set (see the "Alternative definitions" section).

```
lemma (in group3) OrderedGroup_ZF_1_L28:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ∈ G⁺
  shows b≤a
proof -
  from A2 have 1 ≤ a·b⁻¹ using OrderedGroup_ZF_1_L2
    by simp
  with A1 show b≤a using OrderedGroup_ZF_1_L5K
    by simp
qed
```

A special case of `OrderedGroup_ZF_1_L28` when $a \cdot b^{-1}$ is positive.

corollary (**in** group3) OrderedGroup_ZF_1_L29:
  assumes A1: a∈G b∈G **and** A2: a·b⁻¹ ∈ G₊
  shows b≤a b≠a
proof -
  from A2 **have** **1** ≤ a·b⁻¹ **and** I: a·b⁻¹ ≠ **1**
    using OrderedGroup_ZF_1_L2A **by** auto
  with A1 **show** b≤a **using** OrderedGroup_ZF_1_L5K
    by simp
  from A1 I **show** b≠a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6
    by auto
qed

A bit stronger that OrderedGroup_ZF_1_L29, adds case when two elements are equal.

lemma (**in** group3) OrderedGroup_ZF_1_L30:
  assumes a∈G b∈G **and** a=b ∨ b·a⁻¹ ∈ G₊
  shows a≤b
  using assms OrderedGroup_ZF_1_L3 OrderedGroup_ZF_1_L29
  by auto

A different take on decomposition: we can have $a = b$ or $a < b$ or $b < a$.

lemma (**in** group3) OrderedGroup_ZF_1_L31:
  assumes A1: r {is total on} G **and** A2: a∈G b∈G
  shows a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
proof -
  from A2 **have** a·b⁻¹ ∈ G **using** OrderedGroup_ZF_1_L1
    group0.inverse_in_group group0.group_op_closed
    by simp
  with A1 **have** a·b⁻¹ = **1** ∨ a·b⁻¹ ∈ G₊ ∨ (a·b⁻¹)⁻¹ ∈ G₊
    using OrderedGroup_ZF_1_L14 **by** simp
  moreover
  { **assume** a·b⁻¹ = **1**
    then have a·b⁻¹·b = **1**·b **by** simp
    with A2 **have** a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup_ZF_1_L1
 group0.inv_cancel_two group0.group0_2_L2 **by** auto }
  moreover
  { **assume** a·b⁻¹ ∈ G₊
    with A2 **have** a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup_ZF_1_L29 **by** auto }
  moreover
  { **assume** (a·b⁻¹)⁻¹ ∈ G₊
    with A2 **have** b·a⁻¹ ∈ G₊ **using** OrderedGroup_ZF_1_L1
      group0.group0_2_L12 **by** simp
    with A2 **have** a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup_ZF_1_L29 **by** auto }
  ultimately **show** a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
    by auto

qed

40.4 Intervals and bounded sets

Intervals here are the closed intervals of the form $\{x \in G.a \leq x \leq b\}$.

A bounded set can be translated to put it in G^+ and then it is still bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L1:
  assumes A1: ∀g∈A. L≤g ∧ g≤M
  and A2: S = RightTranslation(G,P,L⁻¹)
  and A3: a ∈ S(A)
  shows a ≤ M·L⁻¹   1≤a
proof -
  from A3 have A≠0 using func1_1_L13A by fast
  then obtain g where g∈A by auto
  with A1 have T1: L∈G M∈G L⁻¹∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
    group0.inverse_in_group by auto
  with A2 have S : G→G using OrderedGroup_ZF_1_L1 group0.group0_5_L1
    by simp
  moreover from A1 have T2: A⊆G using OrderedGroup_ZF_1_L4 by auto
  ultimately have S(A) = {S(b). b∈A} using func_imagedef
    by simp
  with A3 obtain b where T3: b∈A a = S(b) by auto
  with A1 ordGroupAssum T1 have b·L⁻¹≤M·L⁻¹ L·L⁻¹≤b·L⁻¹
    using IsAnOrdGroup_def by auto
  with T3 A2 T1 T2 show a≤M·L⁻¹ 1≤a
    using OrderedGroup_ZF_1_L1 group0.group0_5_L2 group0.group0_2_L6
    by auto
qed
```

Every bounded set is an image of a subset of an interval that starts at 1.

```
lemma (in group3) OrderedGroup_ZF_2_L2:
  assumes A1: IsBounded(A,r)
  shows ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
proof -
  { assume A2: A=0
    let B = 0
    let g = 1
    let T = ConstantFunction(G,1)
    have g∈G⁺ using OrderedGroup_ZF_1_L3A by simp
    moreover have T : G→G
      using func1_3_L1 OrderedGroup_ZF_1_L1 group0.group0_2_L2 by simp
    moreover from A2 have A = T(B) by simp
    moreover have B ⊆ Interval(r,1,g) by simp
    ultimately have
      ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
      by auto }
```

```
 moreover
 { assume A3: A≠0
   with A1 have ∃L. ∀x∈A. L≤x and ∃U. ∀x∈A. x≤U
     using IsBounded_def IsBoundedBelow_def IsBoundedAbove_def
     by auto
   then obtain L U where D1: ∀x∈A. L≤x ∧ x≤U
     by auto
   with A3 have T1: A⊆G using OrderedGroup_ZF_1_L4 by auto
   from A3 obtain a where a∈A by auto
   with D1 have T2: L≤a a≤U by auto
   then have T3: L∈G L⁻¹∈ G U∈G
     using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
group0.inverse_in_group by auto
   let T = RightTranslation(G,P,L)
   let B = RightTranslation(G,P,L⁻¹)(A)
   let g = U·L⁻¹
   have g∈G⁺
   proof -
     from T2 have L≤U using Group_order_transitive by fast
     with ordGroupAssum T3 have L·L⁻¹≤g
using IsAnOrdGroup_def by simp
     with T3 show thesis using OrderedGroup_ZF_1_L1 group0.group0_2_L6
OrderedGroup_ZF_1_L2 by simp
   qed
   moreover from T3 have T : G→G
     using OrderedGroup_ZF_1_L1 group0.group0_5_L1
     by simp
   moreover have A = T(B)
   proof -
     from T3 T1 have T(B) = {a·L⁻¹·L. a∈A}
using OrderedGroup_ZF_1_L1 group0.group0_5_L6
by simp
     moreover from T3 T1 have ∀a∈A. a·L⁻¹·L = a·(L⁻¹·L)
using OrderedGroup_ZF_1_L1 group0.group_oper_assoc by auto
     ultimately have T(B) = {a·(L⁻¹·L). a∈A} by simp
     with T3 have T(B) = {a·1. a∈A}
using OrderedGroup_ZF_1_L1 group0.group0_2_L6 by simp
     moreover from T1 have ∀a∈A. a·1=a
using OrderedGroup_ZF_1_L1 group0.group0_2_L2 by auto
     ultimately show thesis by simp
   qed
   moreover have B ⊆ Interval(r,1,g)
   proof
     fix y assume A4: y ∈ B
     let S = RightTranslation(G,P,L⁻¹)
     from D1 have T4: ∀x∈A. L≤x ∧ x≤U by simp
     moreover have T5: S = RightTranslation(G,P,L⁻¹)
by simp
     moreover from A4 have T6: y ∈ S(A) by simp
```

```
      ultimately have y≤U·L⁻¹ using OrderedGroup_ZF_2_L1
 by blast
      moreover from T4 T5 T6 have 1≤y by (rule OrderedGroup_ZF_2_L1)
      ultimately show y ∈ Interval(r,1,g) using Interval_def by auto
    qed
    ultimately have
      ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
      by auto }
  ultimately show thesis by auto
qed
```

If every interval starting at 1 is finite, then every bounded set is finite. I find it interesting that this does not require the group to be linearly ordered (the order to be total).

```
theorem (in group3) OrderedGroup_ZF_2_T1:
  assumes A1: ∀g∈G⁺. Interval(r,1,g) ∈ Fin(G)
  and A2: IsBounded(A,r)
  shows A ∈ Fin(G)
proof -
  from A2 have
    ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
    using OrderedGroup_ZF_2_L2 by simp
  then obtain B g T where D1: g∈G⁺ B ⊆ Interval(r,1,g)
    and D2: T : G→G A = T(B) by auto
  from D1 A1 have B∈Fin(G) using Fin_subset_lemma by blast
  with D2 show thesis using Finite1_L6A by simp
qed
```

In linearly ordered groups finite sets are bounded.

```
theorem (in group3) ord_group_fin_bounded:
  assumes r {is total on} G and B∈Fin(G)
  shows IsBounded(B,r)
  using ordGroupAssum assms IsAnOrdGroup_def IsPartOrder_def Finite_ZF_1_T1
  by simp
```

For nontrivial linearly ordered groups if for every element G we can find one in A that is greater or equal (not necessarily strictly greater), then A can neither be finite nor bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L2A:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  and A3: ∀a∈G. ∃b∈A. a≤b
  shows
  ∀a∈G. ∃b∈A. a≠b ∧ a≤b
  ¬IsBoundedAbove(A,r)
  A ∉ Fin(G)
proof -
  { fix a
    from A1 A2 obtain c where c ∈ G₊
```

```
    using OrderedGroup_ZF_1_L21 by auto
  moreover assume a∈G
  ultimately have
    a·c ∈ G  and I: a < a·c
    using OrderedGroup_ZF_1_L22 by auto
  with A3 obtain b where II: b∈A  and III: a·c ≤ b
    by auto
  moreover from I III have a<b by (rule OrderedGroup_ZF_1_L4A)
  ultimately have ∃b∈A. a≠b ∧ a≤b by auto
} thus ∀a∈G. ∃b∈A. a≠b ∧ a≤b by simp
with ordGroupAssum A1 show
  ¬IsBoundedAbove(A,r)
  A ∉ Fin(G)
  using IsAnOrdGroup_def IsPartOrder_def
  OrderedGroup_ZF_1_L1A Order_ZF_3_L14 Finite_ZF_1_1_L3
  by auto
qed
```

Nontrivial linearly ordered groups are infinite. Recall that `Fin(A)` is the collection of finite subsets of A. In this lemma we show that $G \notin$ `Fin(G)`, that is that G is not a finite subset of itself. This is a way of saying that G is infinite. We also show that for nontrivial linearly ordered groups G_+ is infinite.

```
theorem (in group3) Linord_group_infinite:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  shows
  G₊ ∉ Fin(G)
  G ∉ Fin(G)
proof -
  from A1 A2 show I: G₊ ∉ Fin(G)
    using OrderedGroup_ZF_1_L23 OrderedGroup_ZF_2_L2A
    by simp
  { assume G ∈ Fin(G)
    moreover have G₊ ⊆ G using PositiveSet_def by auto
    ultimately have G₊ ∈ Fin(G) using Fin_subset_lemma
      by blast
    with I have False by simp
  } then show G ∉ Fin(G) by auto
qed
```

A property of nonempty subsets of linearly ordered groups that don't have a maximum: for any element in such subset we can find one that is strictly greater.

```
lemma (in group3) OrderedGroup_ZF_2_L2B:
  assumes A1: r {is total on} G and A2: A⊆G and
  A3: ¬HasAmaximum(r,A) and A4: x∈A
  shows ∃y∈A. x<y
proof -
```

```
  from ordGroupAssum assms have
    antisym(r)
    r {is total on} G
    A⊆G  ¬HasAmaximum(r,A)  x∈A
    using IsAnOrdGroup_def IsPartOrder_def
    by auto
  then have ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x
    using Order_ZF_4_L16 by simp
  then show ∃y∈A. x<y by auto
qed
```

In linearly ordered groups $G \setminus G_+$ is bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L3:
  assumes A1: r {is total on} G shows IsBoundedAbove(G-G+,r)
proof -
  from A1 have ∀a∈G-G+. a≤1
    using OrderedGroup_ZF_1_L17 by simp
  then show IsBoundedAbove(G-G+,r)
    using IsBoundedAbove_def by auto
qed
```

In linearly ordered groups if $A \cap G_+$ is finite, then A is bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L4:
  assumes A1: r {is total on} G and A2: A⊆G
  and A3: A ∩ G+ ∈ Fin(G)
  shows IsBoundedAbove(A,r)
proof -
  have A ∩ (G-G+) ⊆ G-G+ by auto
  with A1 have IsBoundedAbove(A ∩ (G-G+),r)
    using OrderedGroup_ZF_2_L3 Order_ZF_3_L13
    by blast
  moreover from A1 A3 have IsBoundedAbove(A ∩ G+,r)
    using ord_group_fin_bounded IsBounded_def
    by simp
  moreover from A1 ordGroupAssum have
    r {is total on} G  trans(r)  r⊆G×G
    using IsAnOrdGroup_def IsPartOrder_def by auto
  ultimately have IsBoundedAbove(A ∩ (G-G+) ∪ A ∩ G+,r)
    using Order_ZF_3_L3 by simp
  moreover from A2 have A = A ∩ (G-G+) ∪ A ∩ G+
    by auto
  ultimately show  IsBoundedAbove(A,r) by simp
qed
```

If a set $-A \subseteq G$ is bounded above, then A is bounded below.

```
lemma (in group3) OrderedGroup_ZF_2_L5:
  assumes A1: A⊆G and A2: IsBoundedAbove(-A,r)
  shows IsBoundedBelow(A,r)
proof -
```

```
  { assume A = 0 then have IsBoundedBelow(A,r)
      using IsBoundedBelow_def by auto }
  moreover
  { assume A3: A≠0
    from ordGroupAssum have I: GroupInv(G,P) : G→G
      using IsAnOrdGroup_def group0_2_T2 by simp
    with A1 A2 A3 obtain u where D: ∀a∈(-A). a≤u
      using func1_1_L15A IsBoundedAbove_def by auto
    { fix b assume b∈A
      with A1 I D have b⁻¹ ≤ u and T: b∈G
 using func_imagedef by auto
      then have u⁻¹≤(b⁻¹)⁻¹ using OrderedGroup_ZF_1_L5
 by simp
      with T have u⁻¹≤b
 using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
 by simp
    } then have ∀b∈A. ⟨u⁻¹,b⟩ ∈ r by simp
    then have IsBoundedBelow(A,r)
      using Order_ZF_3_L9 by blast }
  ultimately show thesis by auto
qed
```

If $a \leq b$, then the image of the interval $a..b$ by any function is nonempty.

```
lemma (in group3) OrderedGroup_ZF_2_L6:
  assumes a≤b and f:G→G
  shows f(Interval(r,a,b)) ≠ 0
  using ordGroupAssum assms OrderedGroup_ZF_1_L4
    Order_ZF_2_L6 Order_ZF_2_L2A
    IsAnOrdGroup_def IsPartOrder_def func1_1_L15A
  by auto

end
```

41 More on ordered groups

```
theory OrderedGroup_ZF_1 imports OrderedGroup_ZF

begin
```

In this theory we continue the `OrderedGroup_ZF` theory development.

41.1 Absolute value and the triangle inequality

The goal of this section is to prove the triangle inequality for ordered groups.

Absolute value maps G into G.

```
lemma (in group3) OrderedGroup_ZF_3_L1:
  shows AbsoluteValue(G,P,r) : G→G
```

```
proof -
  let f = id(G⁺)
  let g = restrict(GroupInv(G,P),G-G⁺)
  have f : G⁺→G⁺ using id_type by simp
  then have f : G⁺→G using OrderedGroup_ZF_1_L4E fun_weaken_type
    by blast
  moreover have g : G-G⁺→G
  proof -
    from ordGroupAssum have GroupInv(G,P) : G→G
      using IsAnOrdGroup_def group0_2_T2 by simp
    moreover have G-G⁺ ⊆ G by auto
    ultimately show thesis using restrict_type2 by simp
  qed
  moreover have G⁺∩(G-G⁺) = 0 by blast
  ultimately have f ∪ g : G⁺∪(G-G⁺)→G∪G
    by (rule fun_disjoint_Un)
  moreover have G⁺∪(G-G⁺) = G using OrderedGroup_ZF_1_L4E
    by auto
  ultimately show AbsoluteValue(G,P,r) : G→G
    using AbsoluteValue_def by simp
qed
```

If $a \in G^+$, then $|a| = a$.

```
lemma (in group3) OrderedGroup_ZF_3_L2:
  assumes A1: a∈G⁺ shows |a| = a
proof -
  from ordGroupAssum have GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  with A1 show thesis using
    func1_1_L1 OrderedGroup_ZF_1_L4E fun_disjoint_apply1
    AbsoluteValue_def id_conv by simp
qed
```

The absolute value of the unit is the unit. In the additive totation that would be $|0| = 0$.

```
lemma (in group3) OrderedGroup_ZF_3_L2A:
  shows |1| = 1 using OrderedGroup_ZF_1_L3A OrderedGroup_ZF_3_L2
  by simp
```

If a is positive, then $|a| = a$.

```
lemma (in group3) OrderedGroup_ZF_3_L2B:
  assumes a∈G₊ shows |a| = a
  using assms PositiveSet_def Nonnegative_def OrderedGroup_ZF_3_L2
  by auto
```

If $a \in G \setminus G^+$, then $|a| = a^{-1}$.

```
lemma (in group3) OrderedGroup_ZF_3_L3:
   assumes A1: a ∈ G-G⁺ shows |a| = a⁻¹
```

```
proof -
  have domain(id(G+)) = G+
    using id_type func1_1_L1 by auto
  with A1 show thesis using fun_disjoint_apply2 AbsoluteValue_def
    restrict by simp
qed
```

For elements that not greater than the unit, the absolute value is the inverse.

```
lemma (in group3) OrderedGroup_ZF_3_L3A:
  assumes A1: a≤1
  shows |a| = a⁻¹
proof -
  { assume a=1 then have |a| = a⁻¹
      using OrderedGroup_ZF_3_L2A OrderedGroup_ZF_1_L1 group0.group_inv_of_one
      by simp }
  moreover
  { assume a≠1
    with A1 have |a| = a⁻¹ using OrderedGroup_ZF_1_L4C OrderedGroup_ZF_3_L3
      by simp }
  ultimately show |a| = a⁻¹ by blast
qed
```

In linearly ordered groups the absolute value of any element is in G^+.

```
lemma (in group3) OrderedGroup_ZF_3_L3B:
  assumes A1: r {is total on} G and A2: a∈G
  shows |a| ∈ G+
proof -
  { assume a ∈ G+ then have |a| ∈ G+
      using OrderedGroup_ZF_3_L2 by simp }
  moreover
  { assume a ∉ G+
    with A1 A2 have |a| ∈ G+ using OrderedGroup_ZF_3_L3
      OrderedGroup_ZF_1_L6 by simp }
  ultimately show |a| ∈ G+ by blast
qed
```

For linearly ordered groups (where the order is total), the absolute value maps the group into the positive set.

```
lemma (in group3) OrderedGroup_ZF_3_L3C:
  assumes A1: r {is total on} G
  shows AbsoluteValue(G,P,r) : G→G+
proof-
  have AbsoluteValue(G,P,r) : G→G using OrderedGroup_ZF_3_L1
    by simp
  moreover from A1 have T2:
    ∀g∈G. AbsoluteValue(G,P,r)(g)  ∈ G+
    using OrderedGroup_ZF_3_L3B by simp
  ultimately show thesis by (rule func1_1_L1A)
qed
```

If the absolute value is the unit, then the elemnent is the unit.

```
lemma (in group3) OrderedGroup_ZF_3_L3D:
  assumes A1: a∈G and A2: |a| = 1
  shows a = 1
proof -
  { assume a ∈ G+
    with A2 have a = 1 using OrderedGroup_ZF_3_L2 by simp }
  moreover
  { assume a ∉ G+
    with A1 A2 have a = 1 using
      OrderedGroup_ZF_3_L3 OrderedGroup_ZF_1_L1 group0.group0_2_L8A
      by auto }
  ultimately show a = 1 by blast
qed
```

In linearly ordered groups the unit is not greater than the absolute value of any element.

```
lemma (in group3) OrderedGroup_ZF_3_L3E:
  assumes r {is total on} G and a∈G
  shows 1 ≤ |a|
  using assms OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L2 by simp
```

If b is greater than both a and a^{-1}, then b is greater than $|a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L4:
  assumes A1: a≤b and A2: a⁻¹≤ b
  shows |a|≤ b
proof -
  { assume a∈G+
    with A1 have |a|≤ b using OrderedGroup_ZF_3_L2 by simp }
  moreover
  { assume a∉G+
    with A1 A2 have |a|≤ b
      using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_3_L3 by simp }
  ultimately show |a|≤ b by blast
qed
```

In linearly ordered groups $a \leq |a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L5:
  assumes A1: r {is total on} G and A2: a∈G
  shows a ≤ |a|
proof -
  { assume a ∈ G+
    with A2 have a ≤ |a|
      using OrderedGroup_ZF_3_L2 OrderedGroup_ZF_1_L3 by simp }
  moreover
  { assume a ∉ G+
    with A1 A2 have a ≤ |a|
      using OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L4B by simp }
```

```
  ultimately show a ≤ |a| by blast
qed
```

$a^{-1} \leq |a|$ (in additive notation it would be $-a \leq |a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L6:
  assumes A1: a∈G shows a⁻¹ ≤ |a|
proof -
  { assume a ∈ G⁺
    then have T1: 1≤a and T2: |a| = a using OrderedGroup_ZF_1_L2
      OrderedGroup_ZF_3_L2 by auto
    then have a⁻¹≤1⁻¹ using OrderedGroup_ZF_1_L5 by simp
    then have T3: a⁻¹≤1
      using OrderedGroup_ZF_1_L1 group0.group_inv_of_one by simp
    from T3 T1 have a⁻¹≤a by (rule Group_order_transitive)
    with T2 have a⁻¹ ≤ |a| by simp }
  moreover
  { assume A2: a ∉ G⁺
    from A1 have |a| ∈ G
      using OrderedGroup_ZF_3_L1 apply_funtype by auto
    with ordGroupAssum have |a| ≤ |a|
      using IsAnOrdGroup_def IsPartOrder_def refl_def by simp
    with A1 A2 have a⁻¹ ≤ |a| using OrderedGroup_ZF_3_L3 by simp }
  ultimately show a⁻¹ ≤ |a| by blast
qed
```

Some inequalities about the product of two elements of a linearly ordered group and its absolute value.

```
lemma (in group3) OrderedGroup_ZF_3_L6A:
  assumes r {is total on} G and a∈G  b∈G
  shows
  a·b ≤|a|·|b|
  a·b⁻¹ ≤|a|·|b|
  a⁻¹·b ≤|a|·|b|
  a⁻¹·b⁻¹ ≤|a|·|b|
  using assms OrderedGroup_ZF_3_L5 OrderedGroup_ZF_3_L6
    OrderedGroup_ZF_1_L5B by auto
```

$|a^{-1}| \leq |a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L7:
  assumes r {is total on} G and a∈G
  shows |a⁻¹|≤|a|
  using assms OrderedGroup_ZF_3_L5 OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    OrderedGroup_ZF_3_L6 OrderedGroup_ZF_3_L4 by simp
```

$|a^{-1}| = |a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L7A:
  assumes A1: r {is total on} G and A2: a∈G
  shows |a⁻¹| = |a|
```

```
proof -
  from A2 have a⁻¹∈G using OrderedGroup_ZF_1_L1 group0.inverse_in_group
    by simp
  with A1 have |(a⁻¹)⁻¹| ≤ |a⁻¹| using OrderedGroup_ZF_3_L7 by simp
  with A1 A2 have |a⁻¹| ≤ |a|  |a| ≤ |a⁻¹|
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv OrderedGroup_ZF_3_L7
    by auto
  then show thesis by (rule group_order_antisym)
qed
```

$|a \cdot b^{-1}| = |b \cdot a^{-1}|$. It doesn't look so strange in the additive notation: $|a - b| = |b - a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L7B:
  assumes A1: r {is total on} G and A2: a∈G b∈G
  shows |a·b⁻¹| = |b·a⁻¹|
proof -
  from A1 A2 have |(a·b⁻¹)⁻¹| = |a·b⁻¹| using
    OrderedGroup_ZF_1_L1 group0.inverse_in_group group0.group0_2_L1
    monoid0.group0_1_L1 OrderedGroup_ZF_3_L7A by simp
  moreover from A2 have (a·b⁻¹)⁻¹ =  b·a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L12 by simp
  ultimately show thesis by simp
qed
```

Triangle inequality for linearly ordered abelian groups. It would be nice to drop commutativity or give an example that shows we can't do that.

```
theorem (in group3) OrdGroup_triangle_ineq:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G
  shows |a·b| ≤ |a|·|b|
proof -
  from A1 A2 A3 have
    a ≤ |a| b ≤ |b| a⁻¹ ≤ |a| b⁻¹ ≤ |b|
    using OrderedGroup_ZF_3_L5 OrderedGroup_ZF_3_L6 by auto
  then have a·b ≤ |a|·|b| a⁻¹·b⁻¹ ≤ |a|·|b|
    using OrderedGroup_ZF_1_L5B by auto
  with A1 A3 show |a·b| ≤ |a|·|b|
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_two IsCommutative_def

    OrderedGroup_ZF_3_L4 by simp
qed
```

We can multiply the sides of an inequality with absolute value.

```
lemma (in group3) OrderedGroup_ZF_3_L7C:
  assumes P {is commutative on} G
  and r {is total on} G a∈G b∈G
  and |a| ≤ c  |b| ≤ d
  shows |a·b| ≤ c·d
```

```
proof -
  from assms(1,2,3,4) have |a·b| ≤ |a|·|b|
    using OrdGroup_triangle_ineq by simp
  moreover from assms(5,6) have |a|·|b| ≤ c·d
    using OrderedGroup_ZF_1_L5B by simp
  ultimately show thesis by (rule Group_order_transitive)
qed
```

A version of the `OrderedGroup_ZF_3_L7C` but with multiplying by the inverse.

```
lemma (in group3) OrderedGroup_ZF_3_L7CA:
  assumes P {is commutative on} G
  and r {is total on} G and a∈G  b∈G
  and |a| ≤ c  |b| ≤ d
  shows |a·b⁻¹| ≤ c·d
  using assms OrderedGroup_ZF_1_L1 group0.inverse_in_group
  OrderedGroup_ZF_3_L7A OrderedGroup_ZF_3_L7C by simp
```

Triangle inequality with three integers.

```
lemma (in group3) OrdGroup_triangle_ineq3:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G  c∈G
  shows |a·b·c| ≤ |a|·|b|·|c|
proof -
  from A3 have T: a·b ∈ G  |c| ∈ G
    using OrderedGroup_ZF_1_L1 group0.group_op_closed
      OrderedGroup_ZF_3_L1 apply_funtype by auto
  with A1 A2 A3 have |a·b·c| ≤ |a·b|·|c|
    using OrdGroup_triangle_ineq by simp
  moreover from ordGroupAssum A1 A2 A3 T have
    |a·b|·|c| ≤ |a|·|b|·|c|
    using OrdGroup_triangle_ineq IsAnOrdGroup_def by simp
  ultimately show |a·b·c| ≤ |a|·|b|·|c|
    by (rule Group_order_transitive)
qed
```

Some variants of the triangle inequality.

```
lemma (in group3) OrderedGroup_ZF_3_L7D:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G
  and A4: |a·b⁻¹| ≤ c
  shows
  |a| ≤ c·|b|
  |a| ≤ |b|·c
  c⁻¹·a ≤ b
  a·c⁻¹ ≤ b
  a ≤ b·c
proof -
  from A3 A4 have
    T: a·b⁻¹ ∈ G  |b| ∈ G  c∈G  c⁻¹ ∈ G
```

```
    using OrderedGroup_ZF_1_L1
      group0.inverse_in_group group0.group0_2_L1 monoid0.group0_1_L1
      OrderedGroup_ZF_3_L1 apply_funtype  OrderedGroup_ZF_1_L4
    by auto
  from A3 have |a| = |a·b⁻¹·b|
    using OrderedGroup_ZF_1_L1 group0.inv_cancel_two
    by simp
  with A1 A2 A3 T have |a| ≤ |a·b⁻¹|·|b|
    using OrdGroup_triangle_ineq by simp
  with T A4 show |a| ≤ c·|b| using OrderedGroup_ZF_1_L5C
    by blast
  with T A1 show |a| ≤ |b|·c
    using IsCommutative_def by simp
  from A2 T have a·b⁻¹ ≤ |a·b⁻¹|
    using OrderedGroup_ZF_3_L5 by simp
  moreover note A4
  ultimately have I: a·b⁻¹ ≤ c
    by (rule Group_order_transitive)
  with A3 show c⁻¹·a ≤ b
    using OrderedGroup_ZF_1_L5H by simp
  with A1 A3 T show a·c⁻¹ ≤ b
    using IsCommutative_def by simp
  from A1 A3 T I show a ≤ b·c
    using OrderedGroup_ZF_1_L5H IsCommutative_def
    by auto
qed
```

Some more variants of the triangle inequality.

```
lemma (in group3) OrderedGroup_ZF_3_L7E:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G
  and A4: |a·b⁻¹| ≤ c
  shows b·c⁻¹ ≤ a
proof -
  from A3 have a·b⁻¹ ∈ G
    using OrderedGroup_ZF_1_L1
      group0.inverse_in_group group0.group_op_closed
    by auto
  with A2 have |(a·b⁻¹)⁻¹| = |a·b⁻¹|
    using OrderedGroup_ZF_3_L7A by simp
  moreover from A3 have (a·b⁻¹)⁻¹ = b·a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L12
    by simp
  ultimately have |b·a⁻¹| = |a·b⁻¹|
    by simp
  with A1 A2 A3 A4 show b·c⁻¹ ≤ a
    using OrderedGroup_ZF_3_L7D by simp
qed
```

An application of the triangle inequality with four group elements.

```
lemma (in group3) OrderedGroup_ZF_3_L7F:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and
  A3: a∈G  b∈G  c∈G  d∈G
  shows |a·c⁻¹| ≤ |a·b|·|c·d|·|b·d⁻¹|
proof -
  from A3 have T:
    a·c⁻¹ ∈ G  a·b ∈ G  c·d ∈ G  b·d⁻¹ ∈ G
    (c·d)⁻¹ ∈ G  (b·d⁻¹)⁻¹ ∈ G
    using OrderedGroup_ZF_1_L1
      group0.inverse_in_group group0.group_op_closed
    by auto
  with A1 A2 have |(a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹| ≤ |a·b|·|(c·d)⁻¹|·|(b·d⁻¹)⁻¹|
    using OrdGroup_triangle_ineq3 by simp
  moreover from A2 T have |(c·d)⁻¹| =|c·d| and |(b·d⁻¹)⁻¹| = |b·d⁻¹|
    using OrderedGroup_ZF_3_L7A by auto
  moreover from A1 A3 have (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = a·c⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_4_L8
    by simp
  ultimately show |a·c⁻¹| ≤ |a·b|·|c·d|·|b·d⁻¹|
    by simp
qed
```

$|a| \leq L$ implies $L^{-1} \leq a$ (it would be $-L \leq a$ in the additive notation).

```
lemma (in group3) OrderedGroup_ZF_3_L8:
  assumes A1:  a∈G and A2: |a|≤L
   shows
  L⁻¹≤a
proof -
  from A1 have I: a⁻¹ ≤ |a| using OrderedGroup_ZF_3_L6 by simp
  from I A2 have a⁻¹ ≤ L by (rule Group_order_transitive)
  then have L⁻¹≤(a⁻¹)⁻¹ using OrderedGroup_ZF_1_L5 by simp
  with A1 show L⁻¹≤a using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

In linearly ordered groups $|a| \leq L$ implies $a \leq L$ (it would be $a \leq L$ in the additive notation).

```
lemma (in group3) OrderedGroup_ZF_3_L8A:
  assumes A1: r {is total on} G
  and A2: a∈G and A3: |a|≤L
  shows
  a≤L
  1≤L
proof -
  from A1 A2 have I: a ≤ |a| using OrderedGroup_ZF_3_L5 by simp
  from I A3 show a≤L by (rule Group_order_transitive)
  from A1 A2 A3 have 1 ≤ |a|  |a|≤L
     using OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L2 by auto
```

```
    then show 1≤L by (rule Group_order_transitive)
qed
```

A somewhat generalized version of the above lemma.

```
lemma (in group3) OrderedGroup_ZF_3_L8B:
  assumes A1: a∈G and A2: |a|≤L and A3: 1≤c
  shows (L·c)⁻¹ ≤ a
proof -
  from A1 A2 A3 have c⁻¹·L⁻¹ ≤ 1·a
    using OrderedGroup_ZF_3_L8 OrderedGroup_ZF_1_L5AB
    OrderedGroup_ZF_1_L5B by simp
  with A1 A2 A3 show (L·c)⁻¹ ≤ a
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.group_inv_of_two group0.group0_2_L2
    by simp
qed
```

If b is between a and $a \cdot c$, then $b \cdot a^{-1} \leq c$.

```
lemma (in group3) OrderedGroup_ZF_3_L8C:
  assumes A1: a≤b and A2: c∈G and A3: b≤c·a
  shows |b·a⁻¹| ≤ c
proof -
  from A1 A2 A3 have b·a⁻¹ ≤ c
    using OrderedGroup_ZF_1_L9C OrderedGroup_ZF_1_L4
    by simp
  moreover have (b·a⁻¹)⁻¹ ≤ c
  proof -
    from A1 have T: a∈G  b∈G
      using OrderedGroup_ZF_1_L4 by auto
    with A1 have a·b⁻¹ ≤ 1
      using OrderedGroup_ZF_1_L9 by blast
    moreover
    from A1 A3 have a≤c·a
      by (rule Group_order_transitive)
    with ordGroupAssum T have a·a⁻¹ ≤ c·a·a⁻¹
      using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      IsAnOrdGroup_def by simp
    with T A2 have 1 ≤ c
      using OrderedGroup_ZF_1_L1
 group0.group0_2_L6 group0.inv_cancel_two
      by simp
    ultimately have a·b⁻¹ ≤ c
      by (rule Group_order_transitive)
    with T show (b·a⁻¹)⁻¹ ≤ c
      using OrderedGroup_ZF_1_L1 group0.group0_2_L12
      by simp
  qed
  ultimately show |b·a⁻¹| ≤ c
    using OrderedGroup_ZF_3_L4 by simp
```

```
qed
```

For linearly ordered groups if the absolute values of elements in a set are bounded, then the set is bounded.

```
lemma (in group3) OrderedGroup_ZF_3_L9:
  assumes A1: r {is total on} G
  and A2: A⊆G and A3: ∀a∈A. |a| ≤ L
  shows IsBounded(A,r)
proof -
  from A1 A2 A3 have
    ∀a∈A. a≤L  ∀a∈A. L⁻¹≤a
    using OrderedGroup_ZF_3_L8 OrderedGroup_ZF_3_L8A by auto
  then show IsBounded(A,r) using
    IsBoundedAbove_def IsBoundedBelow_def IsBounded_def
    by auto
qed
```

A slightly more general version of the previous lemma, stating the same fact for a set defined by separation.

```
lemma (in group3) OrderedGroup_ZF_3_L9A:
  assumes A1: r {is total on} G
  and A2: ∀x∈X. b(x)∈G ∧ |b(x)|≤L
  shows IsBounded({b(x). x∈X},r)
proof -
  from A2 have {b(x). x∈X} ⊆ G ∀a∈{b(x). x∈X}. |a| ≤ L
    by auto
  with A1 show thesis using OrderedGroup_ZF_3_L9 by blast
qed
```

A special form of the previous lemma stating a similar fact for an image of a set by a function with values in a linearly ordered group.

```
lemma (in group3) OrderedGroup_ZF_3_L9B:
  assumes A1: r {is total on} G
  and A2: f:X→G and A3: A⊆X
  and A4: ∀x∈A. |f(x)| ≤ L
  shows IsBounded(f(A),r)
proof -
  from A2 A3 A4 have ∀x∈A. f(x) ∈ G ∧  |f(x)| ≤ L
    using apply_funtype by auto
  with A1 have  IsBounded({f(x). x∈A},r)
    by (rule OrderedGroup_ZF_3_L9A)
  with A2 A3 show IsBounded(f(A),r)
    using func_imagedef by simp
qed
```

For linearly ordered groups if $l \leq a \leq u$ then $|a|$ is smaller than the greater of $|l|, |u|$.

```
lemma (in group3) OrderedGroup_ZF_3_L10:
```

```
  assumes A1: r {is total on} G
  and A2: l≤a  a≤u
  shows
  |a| ≤ GreaterOf(r,|l|,|u|)
proof -
  from A2 have T1: |l| ∈ G |a| ∈ G |u| ∈ G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_3_L1 apply_funtype
    by auto
  { assume A3: a∈G+
    with A2 have 1≤a a≤u
      using OrderedGroup_ZF_1_L2 by auto
    then have 1≤u by (rule Group_order_transitive)
    with A2 A3 have |a|≤|u|
      using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_3_L2 by simp
    moreover from A1 T1 have |u| ≤ GreaterOf(r,|l|,|u|)
      using Order_ZF_3_L2 by simp
    ultimately have |a| ≤ GreaterOf(r,|l|,|u|)
      by (rule Group_order_transitive) }
  moreover
  { assume A4: a∉G+
    with A2 have T2:
      l∈G |l| ∈ G |a| ∈ G |u| ∈ G a ∈ G-G+
      using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_3_L1 apply_funtype
      by auto
    with A2 have l ∈ G-G+ using OrderedGroup_ZF_1_L4D by fast
    with T2 A2 have |a| ≤ |l|
      using OrderedGroup_ZF_3_L3 OrderedGroup_ZF_1_L5
      by simp
    moreover from A1 T2 have |l| ≤ GreaterOf(r,|l|,|u|)
      using Order_ZF_3_L2 by simp
    ultimately have |a| ≤ GreaterOf(r,|l|,|u|)
      by (rule Group_order_transitive) }
  ultimately show thesis by blast
qed
```

For linearly ordered groups if a set is bounded then the absolute values are bounded.

```
lemma (in group3) OrderedGroup_ZF_3_L10A:
  assumes A1: r {is total on} G
  and A2: IsBounded(A,r)
  shows ∃L. ∀a∈A. |a| ≤ L
proof -
  { assume A = 0 then have thesis by auto }
  moreover
  { assume A3: A≠0
    with A2 have ∃u. ∀g∈A. g≤u and ∃l.∀g∈A. l≤g
      using IsBounded_def IsBoundedAbove_def IsBoundedBelow_def
      by auto
    then obtain u l where ∀g∈A. l≤g ∧  g≤u
```

```
      by auto
    with A1 have ∀a∈A. |a| ≤ GreaterOf(r,|l|,|u|)
      using OrderedGroup_ZF_3_L10 by simp
    then have thesis by auto }
  ultimately show thesis by blast
qed
```

A slightly more general version of the previous lemma, stating the same fact for a set defined by separation.

```
lemma (in group3) OrderedGroup_ZF_3_L11:
  assumes r {is total on} G
  and IsBounded({b(x).x∈X},r)
  shows ∃L. ∀x∈X. |b(x)| ≤ L
  using assms OrderedGroup_ZF_3_L10A by blast
```

Absolute values of elements of a finite image of a nonempty set are bounded by an element of the group.

```
lemma (in group3) OrderedGroup_ZF_3_L11A:
  assumes A1: r {is total on} G
  and A2: X≠0 and A3: {b(x). x∈X} ∈ Fin(G)
  shows ∃L∈G. ∀x∈X. |b(x)| ≤ L
proof -
  from A1 A3 have ∃L. ∀x∈X. |b(x)| ≤ L
     using  ord_group_fin_bounded OrderedGroup_ZF_3_L11
     by simp
  then obtain L where I: ∀x∈X. |b(x)| ≤ L
    using OrderedGroup_ZF_3_L11 by auto
  from A2 obtain x where x∈X by auto
  with I show thesis using OrderedGroup_ZF_1_L4
    by blast
qed
```

In totally ordered groups the absolute value of a nonunit element is in G_+.

```
lemma (in group3) OrderedGroup_ZF_3_L12:
  assumes A1: r {is total on} G
  and A2: a∈G  and A3: a≠1
  shows |a| ∈ G+
proof -
  from A1 A2 have |a| ∈ G  1 ≤ |a|
    using OrderedGroup_ZF_3_L1 apply_funtype
      OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L2
    by auto
  moreover from A2 A3 have |a| ≠ 1
    using OrderedGroup_ZF_3_L3D by auto
  ultimately show |a| ∈ G+
    using PositiveSet_def by auto
qed
```

41.2 Maximum absolute value of a set

Quite often when considering inequalities we prefer to talk about the absolute values instead of raw elements of a set. This section formalizes some material that is useful for that.

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum belongs to the image of the set by the absolute value function.

```
lemma (in group3) OrderedGroup_ZF_4_L1:
  assumes A ⊆ G
  and HasAmaximum(r,A) HasAminimum(r,A)
  and M = GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
  shows M ∈ AbsoluteValue(G,P,r)(A)
  using ordGroupAssum assms IsAnOrdGroup_def IsPartOrder_def
    Order_ZF_4_L3 Order_ZF_4_L4 OrderedGroup_ZF_3_L1
    func_imagedef GreaterOf_def by auto
```

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum bounds absolute values of all elements of the set.

```
lemma (in group3) OrderedGroup_ZF_4_L2:
  assumes A1: r {is total on} G
  and A2: HasAmaximum(r,A)  HasAminimum(r,A)
  and A3: a∈A
  shows |a|≤ GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
proof -
  from ordGroupAssum A2 A3 have
    Minimum(r,A)≤ a a≤ Maximum(r,A)
    using IsAnOrdGroup_def IsPartOrder_def Order_ZF_4_L3 Order_ZF_4_L4
    by auto
  with A1 show thesis by (rule OrderedGroup_ZF_3_L10)
qed
```

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum bounds absolute values of all elements of the set. In this lemma the absolute values of ekements of a set are represented as the elements of the image of the set by the absolute value function.

```
lemma (in group3) OrderedGroup_ZF_4_L3:
  assumes r {is total on} G and A ⊆ G
  and HasAmaximum(r,A) HasAminimum(r,A)
  and b ∈ AbsoluteValue(G,P,r)(A)
  shows b≤ GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
  using assms OrderedGroup_ZF_3_L1 func_imagedef OrderedGroup_ZF_4_L2
  by auto
```

If a set has a maximum and minimum, then the set of absolute values also has a maximum.

```
lemma (in group3) OrderedGroup_ZF_4_L4:
  assumes A1: r {is total on} G and A2: A ⊆ G
  and A3: HasAmaximum(r,A) HasAminimum(r,A)
  shows HasAmaximum(r,AbsoluteValue(G,P,r)(A))
proof -
  let M = GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
  from A2 A3 have M ∈ AbsoluteValue(G,P,r)(A)
    using OrderedGroup_ZF_4_L1 by simp
  moreover from A1 A2 A3 have
    ∀b ∈ AbsoluteValue(G,P,r)(A). b ≤ M
    using OrderedGroup_ZF_4_L3 by simp
  ultimately show thesis using HasAmaximum_def by auto
qed
```

If a set has a maximum and a minimum, then all absolute values are bounded by the maximum of the set of absolute values.

```
lemma (in group3) OrderedGroup_ZF_4_L5:
  assumes A1: r {is total on} G and A2: A ⊆ G
  and A3: HasAmaximum(r,A) HasAminimum(r,A)
  and A4: a∈A
  shows |a| ≤ Maximum(r,AbsoluteValue(G,P,r)(A))
proof -
  from A2 A4 have |a| ∈ AbsoluteValue(G,P,r)(A)
    using OrderedGroup_ZF_3_L1 func_imagedef by auto
  with ordGroupAssum A1 A2 A3 show thesis using
    IsAnOrdGroup_def IsPartOrder_def OrderedGroup_ZF_4_L4
    Order_ZF_4_L3 by simp
qed
```

41.3 Alternative definitions

Sometimes it is usful to define the order by prescibing the set of positive or nonnegative elements. This section deals with two such definitions. One takes a subset H of G that is closed under the group operation, $1 \notin H$ and for every $a \in H$ we have either $a \in H$ or $a^{-1} \in H$. Then the order is defined as $a \leq b$ iff $a = b$ or $a^{-1}b \in H$. For abelian groups this makes a linearly ordered group. We will refer to order defined this way in the comments as the order defined by a positive set. The context used in this section is the `group0` context defined in `Group_ZF` theory. Recall that `f` in that context denotes the group operation (unlike in the previous sections where the group operation was denoted `P`.

The order defined by a positive set is the same as the order defined by a nonnegative set.

```
lemma (in group0) OrderedGroup_ZF_5_L1:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  shows ⟨a,b⟩ ∈ r  ⟷ a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
```

```
proof
  assume ⟨a,b⟩ ∈ r
  with A1 show a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
    using group0_2_L6 by auto
next assume a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
   then have a∈G ∧ b∈G ∧ b=(a⁻¹)⁻¹ ∨  a∈G ∧ b∈G ∧ a⁻¹·b ∈ H
    using  inverse_in_group group0_2_L9 by auto
  with A1 show ⟨a,b⟩ ∈ r using group_inv_of_inv
    by auto
qed
```

The relation defined by a positive set is antisymmetric.

```
lemma (in group0) OrderedGroup_ZF_5_L2:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  and A2: ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  shows antisym(r)
proof -
  { fix a b assume A3: ⟨a,b⟩ ∈ r  ⟨b,a⟩ ∈ r
    with A1 have T: a∈G  b∈G by auto
    { assume A4: a≠b
      with A1 A3 have a⁻¹·b ∈ G  a⁻¹·b ∈ H  (a⁻¹·b)⁻¹ ∈ H
 using inverse_in_group group0_2_L1 monoid0.group0_1_L1 group0_2_L12
 by auto
      with A2 have a⁻¹·b = 1 using Xor_def by auto
      with T A4 have False using group0_2_L11 by auto
    } then have a=b by auto
  } then show antisym(r) by (rule antisymI)
qed
```

The relation defined by a positive set is transitive.

```
lemma (in group0) OrderedGroup_ZF_5_L3:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  and A2: H⊆G  H {is closed under} P
  shows trans(r)
proof -
  { fix a b c assume ⟨a,b⟩ ∈ r  ⟨b,c⟩ ∈ r
    with A1 have
      a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
      b∈G ∧ c∈G ∧ b⁻¹·c ∈ H ∪ {1}
      using OrderedGroup_ZF_5_L1 by auto
    with A2 have
      I: a∈G  b∈G  c∈G
      and (a⁻¹·b)·(b⁻¹·c) ∈  H ∪ {1}
      using inverse_in_group group0_2_L17 IsOpClosed_def
      by auto
    moreover from I have a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
      by (rule group0_2_L14A)
    ultimately have ⟨a,c⟩ ∈ G×G  a⁻¹·c  ∈  H ∪ {1}
      by auto
```

```
    with A1 have ⟨a,c⟩ ∈ r using OrderedGroup_ZF_5_L1
      by auto
  } then have ∀ a b c. ⟨a, b⟩ ∈ r ∧ ⟨b, c⟩ ∈ r ⟶ ⟨a, c⟩ ∈ r
    by blast
  then show  trans(r) by (rule Fol1_L2)
qed
```

The relation defined by a positive set is translation invariant. With our definition this step requires the group to be abelian.

```
lemma (in group0) OrderedGroup_ZF_5_L4:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  and A2: P {is commutative on} G
  and A3: ⟨a,b⟩ ∈ r  and A4: c∈G
  shows ⟨a·c,b·c⟩ ∈ r ∧ ⟨c·a,c·b⟩ ∈ r
proof
  from A1 A3 A4 have
    I: a∈G  b∈G  a·c ∈ G  b·c ∈ G
    and II: a⁻¹·b ∈ H ∪ {1}
    using OrderedGroup_ZF_5_L1 group_op_closed
    by auto
  with A2 A4 have (a·c)⁻¹·(b·c) ∈ H ∪ {1}
    using group0_4_L6D by simp
  with A1 I show ⟨a·c,b·c⟩ ∈ r using  OrderedGroup_ZF_5_L1
    by auto
  with A2 A4 I show ⟨c·a,c·b⟩ ∈ r
    using IsCommutative_def by simp
qed
```

If $H \subseteq G$ is closed under the group operation $1 \notin H$ and for every $a \in H$ we have either $a \in H$ or $a^{-1} \in H$, then the relation "$\leq$" defined by $a \leq b \Leftrightarrow a^{-1}b \in H$ orders the group G. In such order H may be the set of positive or nonnegative elements.

```
lemma (in group0) OrderedGroup_ZF_5_L5:
  assumes A1: P {is commutative on} G
  and A2: H⊆G  H {is closed under} P
  and A3: ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  and A4: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  shows
  IsAnOrdGroup(G,P,r)
  r {is total on} G
  Nonnegative(G,P,r) = PositiveSet(G,P,r) ∪ {1}
proof -
  from groupAssum A2 A3 A4 have
    IsAgroup(G,P)  r ⊆ G×G  IsPartOrder(G,r)
    using refl_def OrderedGroup_ZF_5_L2 OrderedGroup_ZF_5_L3
      IsPartOrder_def by auto
  moreover from A1 A4 have
    ∀g∈G. ∀a b. ⟨ a,b⟩ ∈ r ⟶ ⟨a·g,b·g⟩ ∈ r ∧ ⟨g·a,g·b⟩ ∈ r
    using OrderedGroup_ZF_5_L4 by blast
```

```
  ultimately show IsAnOrdGroup(G,P,r)
    using IsAnOrdGroup_def by simp
  then show Nonnegative(G,P,r) = PositiveSet(G,P,r) ∪ {1}
    using group3_def group3.OrderedGroup_ZF_1_L24
    by simp
  { fix a b
    assume T: a∈G  b∈G
    then have T1: a⁻¹·b ∈ G
      using inverse_in_group group_op_closed by simp
    { assume ⟨ a,b⟩ ∉ r
      with A4 T have I: a≠b and II: a⁻¹·b ∉ H
 by auto
      from A3 T T1 I have (a⁻¹·b ∈ H) Xor ((a⁻¹·b)⁻¹ ∈ H)
 using group0_2_L11 by auto
      with A4 T II have ⟨ b,a⟩ ∈ r
 using Xor_def group0_2_L12 by simp
    } then have ⟨ a,b⟩ ∈ r ∨ ⟨ b,a⟩ ∈ r by auto
  } then show r {is total on} G using IsTotal_def
    by simp
qed
```

If the set defined as in `OrderedGroup_ZF_5_L4` does not contain the neutral element, then it is the positive set for the resulting order.

```
lemma (in group0) OrderedGroup_ZF_5_L6:
  assumes P {is commutative on} G
  and H⊆G and 1 ∉ H
  and r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  shows PositiveSet(G,P,r) = H
  using assms group_inv_of_one group0_2_L2 PositiveSet_def
  by auto
```

The next definition describes how we construct an order relation from the prescribed set of positive elements.

```
definition
  OrderFromPosSet(G,P,H) ≡
  {p ∈ G×G. fst(p) = snd(p) ∨ P⟨GroupInv(G,P)(fst(p)),snd(p)⟩ ∈ H }
```

The next theorem rephrases lemmas `OrderedGroup_ZF_5_L5` and `OrderedGroup_ZF_5_L6` using the definition of the order from the positive set `OrderFromPosSet`. To summarize, this is what it says: Suppose that $H \subseteq G$ is a set closed under that group operation such that $1 \notin H$ and for every nonunit group element a either $a \in H$ or $a^{-1} \in H$. Define the order as $a \leq b$ iff $a = b$ or $a^{-1} \cdot b \in H$. Then this order makes G into a linearly ordered group such H is the set of positive elements (and then of course $H \cup \{1\}$ is the set of nonnegative elements).

```
theorem (in group0) Group_ord_by_positive_set:
  assumes P {is commutative on} G
  and H⊆G   H {is closed under} P   1 ∉ H
```

```
  and ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  shows
  IsAnOrdGroup(G,P,OrderFromPosSet(G,P,H))
  OrderFromPosSet(G,P,H) {is total on} G
  PositiveSet(G,P,OrderFromPosSet(G,P,H)) = H
  Nonnegative(G,P,OrderFromPosSet(G,P,H)) = H ∪ {1}
  using assms OrderFromPosSet_def OrderedGroup_ZF_5_L5 OrderedGroup_ZF_5_L6
  by auto
```

41.4 Odd Extensions

In this section we verify properties of odd extensions of functions defined on G_+. An odd extension of a function $f : G_+ \to G$ is a function $f^\circ : G \to G$ defined by $f^\circ(x) = f(x)$ if $x \in G_+$, $f(1) = 1$ and $f^\circ(x) = (f(x^{-1}))^{-1}$ for $x < 1$. Such function is the unique odd function that is equal to f when restricted to G_+.

The next lemma is just to see the definition of the odd extension in the notation used in the group1 context.

```
lemma (in group3) OrderedGroup_ZF_6_L1:
  shows f° = f ∪ {⟨a, (f(a⁻¹))⁻¹⟩. a ∈ -G₊} ∪ {⟨1,1⟩}
  using OddExtension_def by simp
```

A technical lemma that states that from a function defined on G_+ with values in G we have $(f(a^{-1}))^{-1} \in G$.

```
lemma (in group3) OrderedGroup_ZF_6_L2:
  assumes f: G₊→G and a∈-G₊
  shows
  f(a⁻¹) ∈ G
  (f(a⁻¹))⁻¹ ∈ G
  using assms OrderedGroup_ZF_1_L27 apply_funtype
    OrderedGroup_ZF_1_L1 group0.inverse_in_group
  by auto
```

The main theorem about odd extensions. It basically says that the odd extension of a function is what we want to to be.

```
lemma (in group3) odd_ext_props:
  assumes A1: r {is total on} G and A2: f: G₊→G
  shows
  f° : G → G
  ∀a∈G₊. (f°)(a) = f(a)
  ∀a∈(-G₊). (f°)(a) = (f(a⁻¹))⁻¹
  (f°)(1) = 1
proof -
  from A1 A2 have I:
    f: G₊→G
    ∀a∈-G₊. (f(a⁻¹))⁻¹ ∈ G
    G₊∩(-G₊) = 0
```

```
    1 ∉ G₊∪(-G₊)
    f° = f ∪ {⟨a, (f(a⁻¹))⁻¹⟩. a ∈ -G₊} ∪ {⟨1,1⟩}
    using OrderedGroup_ZF_6_L2 OrdGroup_decomp2 OrderedGroup_ZF_6_L1
    by auto
  then have f°: G₊ ∪ (-G₊) ∪ {1} →G∪G∪{1}
    by (rule func1_1_L11E)
  moreover from A1 have
    G₊ ∪ (-G₊) ∪ {1} = G
    G∪G∪{1} = G
    using OrdGroup_decomp2 OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  ultimately show f° : G → G by simp
  from I show ∀a∈G₊. (f°)(a) = f(a)
    by (rule func1_1_L11E)
  from I show ∀a∈(-G₊). (f°)(a) = (f(a⁻¹))⁻¹
    by (rule func1_1_L11E)
  from I show (f°)(1) = 1
    by (rule func1_1_L11E)
qed
```

Odd extensions are odd, of course.

```
lemma (in group3) oddext_is_odd:
  assumes A1: r {is total on} G and A2: f: G₊→G
  and A3: a∈G
  shows (f°)(a⁻¹) = ((f°)(a))⁻¹
proof -
  from A1 A3 have a∈G₊ ∨ a ∈ (-G₊) ∨ a=1
    using OrdGroup_decomp2 by blast
  moreover
  { assume a∈G₊
    with A1 A2 have a⁻¹ ∈ -G₊ and  (f°)(a) = f(a)
      using OrderedGroup_ZF_1_L25 odd_ext_props by auto
    with A1 A2 have
      (f°)(a⁻¹) = (f((a⁻¹)⁻¹))⁻¹  and (f(a))⁻¹ = ((f°)(a))⁻¹
      using odd_ext_props by auto
    with A3 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp }
  moreover
  { assume A4: a ∈ -G₊
    with A1 A2  have a⁻¹ ∈ G₊ and  (f°)(a) = (f(a⁻¹))⁻¹
      using OrderedGroup_ZF_1_L27 odd_ext_props
      by auto
    with A1 A2 A4 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using odd_ext_props OrderedGroup_ZF_6_L2
 OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp }
  moreover
  { assume a = 1
```

```
    with A1 A2 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using OrderedGroup_ZF_1_L1 group0.group_inv_of_one
 odd_ext_props by simp
  }
  ultimately show (f°)(a⁻¹) = ((f°)(a))⁻¹
    by auto
qed
```

Another way of saying that odd extensions are odd.

```
lemma (in group3) oddext_is_odd_alt:
  assumes A1: r {is total on} G and A2: f: G₊→G
  and A3: a∈G
  shows ((f°)(a⁻¹))⁻¹ = (f°)(a)
proof -
  from A1 A2 have
    f° : G → G
    ∀a∈G. (f°)(a⁻¹) = ((f°)(a))⁻¹
    using odd_ext_props oddext_is_odd by auto
  then have ∀a∈G. ((f°)(a⁻¹))⁻¹ = (f°)(a)
    using OrderedGroup_ZF_1_L1 group0.group0_6_L2 by simp
  with A3 show ((f°)(a⁻¹))⁻¹ = (f°)(a) by simp
qed
```

41.5 Functions with infinite limits

In this section we consider functions $f : G \to G$ with the property that for $f(x)$ is arbitrarily large for large enough x. More precisely, for every $a \in G$ there exist $b \in G_+$ such that for every $x \geq b$ we have $f(x) \geq a$. In a sense this means that $\lim_{x\to\infty} f(x) = \infty$, hence the title of this section. We also prove dual statements for functions such that $\lim_{x\to-\infty} f(x) = -\infty$.

If an image of a set by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in group3) OrderedGroup_ZF_7_L1:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: f:G→G and
  A4: ∀a∈G.∃b∈G₊.∀x. b≤x ⟶ a ≤ f(x) and
  A5: A⊆G and
  A6: IsBoundedAbove(f(A),r)
  shows IsBoundedAbove(A,r)
proof -
  { assume ¬IsBoundedAbove(A,r)
    then have I: ∀u. ∃x∈A. ¬(x≤u)
      using IsBoundedAbove_def by auto
    have ∀a∈G. ∃y∈f(A). a≤y
    proof -
      { fix a assume a∈G
 with A4 obtain b where
```

```
    II: b∈G+ and III: ∀x. b≤x ⟶ a ≤ f(x)
    by auto
  from I obtain x where IV: x∈A and ¬(x≤b)
    by auto
  with A1 A5 II have
    r {is total on} G
    x∈G  b∈G  ¬(x≤b)
    using PositiveSet_def by auto
  with III have a ≤ f(x)
    using OrderedGroup_ZF_1_L8 by blast
  with A3 A5 IV have ∃y∈f(A). a≤y
    using func_imagedef by auto
      } thus thesis by simp
     qed
     with A1 A2 A6 have False using OrderedGroup_ZF_2_L2A
       by simp
   } thus thesis by auto
qed
```

If an image of a set defined by separation by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in group3) OrderedGroup_ZF_7_L2:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G+.∀y. b≤y ⟶ a ≤ f(y) and
  A6: ∀x∈X. b(x) ∈ G ∧ f(b(x)) ≤ U
  shows ∃u.∀x∈X. b(x) ≤ u
proof -
  let A = {b(x). x∈X}
  from A6 have I: A⊆G by auto
  moreover note assms
  moreover have IsBoundedAbove(f(A),r)
  proof -
    from A4 A6 I have ∀z∈f(A). ⟨z,U⟩ ∈ r
      using func_imagedef by simp
    then show IsBoundedAbove(f(A),r)
      by (rule Order_ZF_3_L10)
  qed
  ultimately have IsBoundedAbove(A,r) using OrderedGroup_ZF_7_L1
    by simp
  with A3 have ∃u.∀y∈A. y ≤ u
    using IsBoundedAbove_def by simp
  then show ∃u.∀x∈X. b(x) ≤ u by auto
qed
```

If the image of a set defined by separation by a function with infinite negative limit is bounded below, then the set itself is bounded above. This is dual to OrderedGroup_ZF_7_L2.

```
lemma (in group3) OrderedGroup_ZF_7_L3:
```

```
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G+.∀y. b≤y ⟶ f(y⁻¹) ≤ a and
  A6: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x))
  shows ∃l.∀x∈X. l ≤ b(x)
proof -
  let g = GroupInv(G,P) O f O GroupInv(G,P)
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  with A4 have II: ∀x∈G. g(x) = (f(x⁻¹))⁻¹
    using func1_1_L18 by simp
  note A1 A2 A3
  moreover from A4 I have g : G→G
    using comp_fun by blast
  moreover have ∀a∈G.∃b∈G+.∀y. b≤y ⟶ a ≤ g(y)
  proof -
  { fix a assume A7: a∈G
    then have a⁻¹ ∈ G
      using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      by simp
    with A5 obtain b where
      III: b∈G+ and ∀y. b≤y ⟶ f(y⁻¹) ≤ a⁻¹
      by auto
    with II A7 have ∀y. b≤y ⟶ a ≤ g(y)
      using OrderedGroup_ZF_1_L5AD OrderedGroup_ZF_1_L4
      by simp
    with III have ∃b∈G+.∀y. b≤y ⟶ a ≤ g(y)
      by auto
  } then show ∀a∈G.∃b∈G+.∀y. b≤y ⟶ a ≤ g(y)
    by simp
  qed
  moreover have ∀x∈X. b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
  proof-
    { fix x assume x∈X
      with A6 have
 T: b(x) ∈ G  b(x)⁻¹ ∈ G and L ≤ f(b(x))
 using OrderedGroup_ZF_1_L1 group0.inverse_in_group
 by auto
      then have (f(b(x)))⁻¹ ≤ L⁻¹
 using OrderedGroup_ZF_1_L5 by simp
      moreover from II T have (f(b(x)))⁻¹ =  g(b(x)⁻¹)
 using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
 by simp
      ultimately have g(b(x)⁻¹) ≤ L⁻¹ by simp
      with T have b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
 by simp
    } then show ∀x∈X. b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
      by simp
  qed
```

```
  ultimately have ∃u.∀x∈X. (b(x))⁻¹ ≤ u
    by (rule OrderedGroup_ZF_7_L2)
  then have ∃u.∀x∈X. u⁻¹ ≤ (b(x)⁻¹)⁻¹
    using OrderedGroup_ZF_1_L5 by auto
  with A6 show ∃l.∀x∈X. l ≤ b(x)
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by auto
qed
```

The next lemma combines OrderedGroup_ZF_7_L2 and OrderedGroup_ZF_7_L3 to show that if an image of a set defined by separation by a function with infinite limits is bounded, then the set itself i bounded.

```
lemma (in group3) OrderedGroup_ZF_7_L4:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ a ≤ f(y) and
  A6: ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ f(y⁻¹) ≤ a and
  A7: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x)) ∧ f(b(x)) ≤ U
shows ∃M.∀x∈X. |b(x)| ≤ M
proof -
  from A7 have
    I: ∀x∈X. b(x) ∈ G ∧ f(b(x)) ≤ U and
    II: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x))
    by auto
  from A1 A2 A3 A4 A5 I have ∃u.∀x∈X. b(x) ≤ u
    by (rule OrderedGroup_ZF_7_L2)
  moreover from  A1 A2 A3 A4 A6 II have ∃l.∀x∈X. l ≤ b(x)
    by (rule OrderedGroup_ZF_7_L3)
  ultimately have ∃u l. ∀x∈X. l≤b(x) ∧ b(x) ≤ u
    by auto
  with A1 have ∃u l.∀x∈X. |b(x)| ≤ GreaterOf(r,|l|,|u|)
    using OrderedGroup_ZF_3_L10 by blast
  then show ∃M.∀x∈X. |b(x)| ≤ M
    by auto
qed

end
```

42 Rings - introduction

```
theory Ring_ZF imports AbelianGroup_ZF

begin
```

This theory file covers basic facts about rings.

42.1 Definition and basic properties

In this section we define what is a ring and list the basic properties of rings.

We say that three sets (R, A, M) form a ring if (R, A) is an abelian group, (R, M) is a monoid and A is distributive with respect to M on R. A represents the additive operation on R. As such it is a subset of $(R \times R) \times R$ (recall that in ZF set theory functions are sets). Similarly M represents the multiplicative operation on R and is also a subset of $(R \times R) \times R$. We don't require the multiplicative operation to be commutative in the definition of a ring.

definition
```
IsAring(R,A,M) ≡ IsAgroup(R,A) ∧ (A {is commutative on} R) ∧
IsAmonoid(R,M) ∧ IsDistributive(R,A,M)
```

We also define the notion of having no zero divisors. In standard notation the ring has no zero divisors if for all $a, b \in R$ we have $a \cdot b = 0$ implies $a = 0$ or $b = 0$.

definition
```
HasNoZeroDivs(R,A,M) ≡ (∀a∈R. ∀b∈R.
M⟨ a,b⟩ = TheNeutralElement(R,A) ⟶
a = TheNeutralElement(R,A) ∨ b = TheNeutralElement(R,A))
```

Next we define a locale that will be used when considering rings.

locale ring0 =

```
fixes R and A and M

assumes ringAssum: IsAring(R,A,M)

fixes ringa (infixl + 90)
defines ringa_def [simp]: x+y ≡ A⟨x,y⟩

fixes ringminus (- _ 89)
defines ringminus_def [simp]: (-x) ≡ GroupInv(R,A)(x)

fixes ringsub (infixl - 90)
defines ringsub_def [simp]: x-y ≡ x+(-y)

fixes ringm (infixl · 95)
defines ringm_def [simp]: x·y ≡ M⟨x,y⟩

fixes ringzero (0)
defines ringzero_def [simp]: 0 ≡ TheNeutralElement(R,A)

fixes ringone (1)
defines ringone_def [simp]: 1 ≡ TheNeutralElement(R,M)
```

```
  fixes ringtwo (2)
  defines ringtwo_def [simp]: 2 ≡ 1+1

  fixes ringsq (_² [96] 97)
  defines ringsq_def [simp]: x² ≡ x·x

  fixes rlistsum (∑ _ 70)
  defines rlistsum_def [simp]: ∑s ≡ Fold(A,0,s)

  fixes rnat_mult (infix · 95)
  defines nat_mult_def [simp]: n·x ≡ ∑{⟨k,x⟩. k∈n}
```

In the ring0 context we can use theorems proven in some other contexts.

```
lemma (in ring0) Ring_ZF_1_L1: shows
  monoid0(R,M)
  group0(R,A)
  A {is commutative on} R
  using ringAssum IsAring_def group0_def monoid0_def by auto
```

The theorems proven in in group0 context (locale) are valid in the ring0 context when applied to the additive group of the ring.

```
sublocale ring0 < add_group: group0 R A ringzero ringa ringminus rlistsum
rnat_mult
  using Ring_ZF_1_L1(2) unfolding ringa_def ringminus_def ringzero_def
by auto
```

The theorem proven in the monoid0 context are valid in the ring0 context when applied to the multiplicative monoid of the ring.

```
sublocale ring0 < mult_monoid: monoid0 R M ringm
  using Ring_ZF_1_L1(1) unfolding ringm_def by auto
```

The additive operation in a ring is distributive with respect to the multiplicative operation.

```
lemma (in ring0) ring_oper_distr: assumes A1: a∈R  b∈R  c∈R
  shows
  a·(b+c) = a·b + a·c
  (b+c)·a = b·a + c·a
  using ringAssum assms IsAring_def IsDistributive_def by auto
```

Zero and one of the ring are elements of the ring. The negative of zero is zero.

```
lemma (in ring0) Ring_ZF_1_L2:
  shows 0∈R  1∈R  (-0) = 0 2∈R
  using add_group.group0_2_L2 mult_monoid.unit_is_neutral
    add_group.group_inv_of_one add_group.group_op_closed by auto
```

The next lemma lists some properties of a ring that require one element of a ring.

```
lemma (in ring0) Ring_ZF_1_L3: assumes a∈R
  shows
  (-a) ∈ R
  (-(-a)) = a
  a+0 = a
  0+a = a
  a·1 = a
  1·a = a
  a-a = 0
  a-0 = a
  2·a = a+a
  (-a)+a = 0
  using assms add_group.inverse_in_group add_group.group_inv_of_inv
    add_group.group0_2_L6 add_group.group0_2_L2 mult_monoid.unit_is_neutral

    Ring_ZF_1_L2 ring_oper_distr
  by auto
```

Properties that require two elements of a ring.

```
lemma (in ring0) Ring_ZF_1_L4: assumes A1: a∈R b∈R
  shows
  a+b ∈ R
  a-b ∈ R
  a·b ∈ R
  a+b = b+a
  using assms Ring_ZF_1_L1(3) Ring_ZF_1_L3
    add_group.monoid.group0_1_L1
    mult_monoid.group0_1_L1
    unfolding IsCommutative_def
  by auto
```

Cancellation of an element on both sides of equality. This is a property of groups, written in the (additive) notation we use for the additive operation in rings.

```
lemma (in ring0) ring_cancel_add:
  assumes A1: a∈R b∈R and A2: a + b = a
  shows b = 0
  using assms add_group.group0_2_L7 by simp
```

Any element of a ring multiplied by zero is zero.

```
lemma (in ring0) Ring_ZF_1_L6:
  assumes A1: x∈R shows 0·x = 0   x·0 = 0
proof -
  let a = x·1
  let b = x·0
  let c = 1·x
  let d = 0·x
  from A1 have
```

```
    a + b = x·(1 + 0)   c + d = (1 + 0)·x
    using Ring_ZF_1_L2 ring_oper_distr by auto
  moreover have x·(1 + 0) = a (1 + 0)·x = c
    using Ring_ZF_1_L2 Ring_ZF_1_L3 by auto
  ultimately have a + b = a and T1: c + d = c
    by auto
  moreover from A1 have
    a ∈ R  b ∈ R and T2: c ∈ R  d ∈ R
    using Ring_ZF_1_L2 Ring_ZF_1_L4 by auto
  ultimately have b = 0 using ring_cancel_add
    by blast
  moreover from T2 T1 have d = 0 using ring_cancel_add
    by blast
  ultimately show x·0 = 0  0·x = 0 by auto
qed
```

Negative can be pulled out of a product.

```
lemma (in ring0) Ring_ZF_1_L7:
  assumes A1: a∈R  b∈R
  shows
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  (-a)·b = a·(-b)
proof -
  from A1 have I:
    a·b ∈ R (-a) ∈ R ((-a)·b) ∈ R
    (-b) ∈ R a·(-b) ∈ R
    using Ring_ZF_1_L3 Ring_ZF_1_L4 by auto
  moreover have (-a)·b + a·b = 0
    and II: a·(-b) + a·b = 0
  proof -
    from A1 I have
      (-a)·b + a·b = ((-a)+ a)·b
      a·(-b) + a·b= a·((-b)+b)
      using ring_oper_distr by auto
    moreover from A1 have
      ((-a)+ a)·b = 0
      a·((-b)+b) = 0
      using add_group.group0_2_L6 Ring_ZF_1_L6
      by auto
    ultimately show
      (-a)·b + a·b = 0
      a·(-b) + a·b = 0
      by auto
  qed
  ultimately show (-a)·b = -(a·b)
    using add_group.group0_2_L9 by simp
  moreover from I II show a·(-b) = -(a·b)
    using add_group.group0_2_L9 by simp
```

```
  ultimately show (-a)·b = a·(-b) by simp
qed
```

Minus times minus is plus.

```
lemma (in ring0) Ring_ZF_1_L7A: assumes a∈R  b∈R
  shows (-a)·(-b) = a·b
  using assms Ring_ZF_1_L3 Ring_ZF_1_L7 Ring_ZF_1_L4
  by simp
```

Subtraction is distributive with respect to multiplication.

```
lemma (in ring0) Ring_ZF_1_L8: assumes a∈R  b∈R  c∈R
  shows
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using assms Ring_ZF_1_L3 ring_oper_distr Ring_ZF_1_L7 Ring_ZF_1_L4
  by auto
```

Other basic properties involving two elements of a ring.

```
lemma (in ring0) Ring_ZF_1_L9: assumes a∈R  b∈R
  shows
  (-b)-a = (-a)-b
  (-(a+b)) = (-a)-b
  (-(a-b)) = ((-a)+b)
  a-(-b) = a+b
  using assms Ring_ZF_1_L1(3) add_group.group0_4_L4 add_group.group_inv_of_inv
  by auto
```

If the difference of two element is zero, then those elements are equal.

```
lemma (in ring0) Ring_ZF_1_L9A:
  assumes A1: a∈R  b∈R and A2: a-b = 0
  shows a=b using add_group.group0_2_L11A assms by auto
```

Other basic properties involving three elements of a ring.

```
lemma (in ring0) Ring_ZF_1_L10:
  assumes a∈R  b∈R  c∈R
  shows
  a+(b+c) = a+b+c

  a-(b+c) = a-b-c
  a-(b-c) = a-b+c
  using assms Ring_ZF_1_L1(3) add_group.group_oper_assoc
    add_group.group0_4_L4A by auto
```

Another property with three elements.

```
lemma (in ring0) Ring_ZF_1_L10A:
  assumes A1: a∈R  b∈R  c∈R
  shows a+(b-c) = a+b-c
  using assms Ring_ZF_1_L3 Ring_ZF_1_L10 by simp
```

Associativity of addition and multiplication.

```
lemma (in ring0) Ring_ZF_1_L11:
  assumes a∈R  b∈R  c∈R
  shows
  a+b+c = a+(b+c)
  a·b·c = a·(b·c)
  using assms add_group.group_oper_assoc mult_monoid.sum_associative
  by auto
```

An interpretation of what it means that a ring has no zero divisors.

```
lemma (in ring0) Ring_ZF_1_L12:
  assumes HasNoZeroDivs(R,A,M)
  and a∈R  a≠0  b∈R  b≠0
  shows a·b≠0
  using assms HasNoZeroDivs_def by auto
```

In rings with no zero divisors we can cancel nonzero factors.

```
lemma (in ring0) Ring_ZF_1_L12A:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a∈R  b∈R  c∈R
  and A3: a·c = b·c and A4: c≠0
  shows a=b
proof -
  from A2 have T: a·c ∈ R  a-b ∈ R
    using Ring_ZF_1_L4 by auto
  with A1 A2 A3 have a-b = 0 ∨ c=0
    using Ring_ZF_1_L3 Ring_ZF_1_L8 HasNoZeroDivs_def
    by simp
  with A2 A4 have a∈R  b∈R  a-b = 0
    by auto
  then show a=b by (rule Ring_ZF_1_L9A)
qed
```

In rings with no zero divisors if two elements are different, then after multiplying by a nonzero element they are still different.

```
lemma (in ring0) Ring_ZF_1_L12B:
  assumes A1: HasNoZeroDivs(R,A,M)
  a∈R   b∈R   c∈R   a≠b   c≠0
  shows  a·c ≠ b·c
  using A1 Ring_ZF_1_L12A by auto
```

In rings with no zero divisors multiplying a nonzero element by a nonone element changes the value.

```
lemma (in ring0) Ring_ZF_1_L12C:
  assumes A1: HasNoZeroDivs(R,A,M) and
  A2: a∈R  b∈R and A3: 0≠a  1≠b
  shows a ≠ a·b
proof -
  { assume a = a·b
```

```
    with A1 A2 have a = 0 ∨ b-1 = 0
      using Ring_ZF_1_L3 Ring_ZF_1_L2 Ring_ZF_1_L8
 Ring_ZF_1_L3 Ring_ZF_1_L2 Ring_ZF_1_L4 HasNoZeroDivs_def
      by simp
    with A2 A3 have False
      using Ring_ZF_1_L2 Ring_ZF_1_L9A by auto
  } then show a ≠ a·b by auto
qed
```

If a square is nonzero, then the element is nonzero.

```
lemma (in ring0) Ring_ZF_1_L13:
  assumes a∈R  and a² ≠ 0
  shows a≠0
  using assms Ring_ZF_1_L2 Ring_ZF_1_L6 by auto
```

Square of an element and its opposite are the same.

```
lemma (in ring0) Ring_ZF_1_L14:
  assumes a∈R shows (-a)² = (a²)
  using assms Ring_ZF_1_L7A by simp
```

Adding zero to a set that is closed under addition results in a set that is also closed under addition. This is a property of groups.

```
lemma (in ring0) Ring_ZF_1_L15:
  assumes H ⊆ R and H {is closed under} A
  shows (H ∪ {0}) {is closed under} A
  using assms add_group.group0_2_L17 by simp
```

Adding zero to a set that is closed under multiplication results in a set that is also closed under multiplication.

```
lemma (in ring0) Ring_ZF_1_L16:
  assumes A1: H ⊆ R and A2: H {is closed under} M
  shows (H ∪ {0}) {is closed under} M
  using assms Ring_ZF_1_L2 Ring_ZF_1_L6 IsOpClosed_def
  by auto
```

The ring is trivial iff $0 = 1$.

```
lemma (in ring0) Ring_ZF_1_L17: shows R = {0} ⟷ 0=1
proof
  assume R = {0}
  then show 0=1 using Ring_ZF_1_L2
    by blast
next assume A1: 0 = 1
  then have R ⊆ {0}
    using Ring_ZF_1_L3 Ring_ZF_1_L6 by auto
  moreover have {0} ⊆ R using Ring_ZF_1_L2 by auto
  ultimately show R = {0} by auto
qed
```

The sets $\{m \cdot x : x \in R\}$ and $\{-m \cdot x : x \in R\}$ are the same.

```
lemma (in ring0) Ring_ZF_1_L18: assumes A1: m∈R
  shows {m·x. x∈R} = {(-m)·x. x∈R}
proof
  { fix a assume a ∈ {m·x. x∈R}
    then obtain x where x∈R and a = m·x
      by auto
    with A1 have (-x) ∈ R  and a = (-m)·(-x)
      using Ring_ZF_1_L3 Ring_ZF_1_L7A by auto
    then have a ∈ {(-m)·x. x∈R}
      by auto
  } then show {m·x. x∈R} ⊆ {(-m)·x. x∈R}
    by auto
next
  { fix a assume a ∈ {(-m)·x. x∈R}
    then obtain x where x∈R and a = (-m)·x
      by auto
    with A1 have (-x) ∈ R and a = m·(-x)
      using Ring_ZF_1_L3 Ring_ZF_1_L7 by auto
    then have a ∈ {m·x. x∈R} by auto
  } then show {(-m)·x. x∈R} ⊆ {m·x. x∈R}
    by auto
qed
```

42.2 Rearrangement lemmas

In happens quite often that we want to show a fact like $(a + b)c + d = (ac + d - e) + (bc + e)$in rings. This is trivial in romantic math and probably there is a way to make it trivial in formalized math. However, I don't know any other way than to tediously prove each such rearrangement when it is needed. This section collects facts of this type.

Rearrangements with two elements of a ring.

```
lemma (in ring0) Ring_ZF_2_L1: assumes a∈R b∈R
  shows a+b·a = (b+1)·a
  using assms Ring_ZF_1_L2 ring_oper_distr Ring_ZF_1_L3 Ring_ZF_1_L4
  by simp
```

Rearrangements with two elements and cancelling.

```
lemma (in ring0) Ring_ZF_2_L1A: assumes a∈R b∈R
  shows
  a-b+b = a
  a+b-a = b
  (-a)+b+a = b
  (-a)+(b+a) = b
  a+(b-a) = b
  using assms add_group.inv_cancel_two add_group.group0_4_L6A
    Ring_ZF_1_L1(3) by auto
```

In rings $a-(b+1)c = (a-d-c)+(d-bc)$ and $a+b+(c+d) = a+(b+c)+d$.

```
lemma (in ring0) Ring_ZF_2_L2:
  assumes a∈R  b∈R  c∈R  d∈R
  shows
    a-(b+1)·c = (a-d-c)+(d-b·c)
    a+b+(c+d) = a+b+c+d
    a+b+(c+d) = a+(b+c)+d
proof -
  let B = b·c
  from ringAssum assms have
    A {is commutative on} R and a∈R B ∈ R c∈R d∈R
    unfolding IsAring_def using Ring_ZF_1_L4 by auto
  then have
    a+(-B+c) = a+(-d)+(-c)+(d+(-B))
    by (rule add_group.group0_4_L8)
  with assms show a-(b+1)·c = (a-d-c)+(d-b·c)
    using Ring_ZF_1_L2 ring_oper_distr Ring_ZF_1_L3 by simp
  from assms show a+b+(c+d) = a+b+c+d
    using Ring_ZF_1_L4(1) Ring_ZF_1_L10(1) by simp
  with assms(1,2,3) show a+b+(c+d) = a+(b+c)+d
    using Ring_ZF_1_L10(1) by simp
qed
```

Rerrangement about adding linear functions.

```
lemma (in ring0) Ring_ZF_2_L3:
  assumes A1: a∈R  b∈R  c∈R  d∈R  x∈R
  shows (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
proof -
  from A1 have
    A {is commutative on} R
    a·x ∈ R  b∈R  c·x ∈ R  d∈R
    using Ring_ZF_1_L1 Ring_ZF_1_L4 by auto
  then have A⟨A⟨ a·x,b⟩,A⟨ c·x,d⟩⟩ = A⟨A⟨ a·x,c·x⟩,A⟨ b,d⟩⟩
    using add_group.group0_4_L8(3) by auto
  with A1 show
    (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
    using ring_oper_distr by simp
qed
```

Rearrangement with three elements

```
lemma (in ring0) Ring_ZF_2_L4:
  assumes M {is commutative on} R
  and a∈R  b∈R  c∈R
  shows a·(b·c) = a·c·b and a·b·c = a·c·b
  using assms IsCommutative_def Ring_ZF_1_L11
  by simp_all
```

Some other rearrangements with three elements.

```
lemma (in ring0) ring_rearr_3_elemA:
```

```
  assumes A1: M {is commutative on} R and
  A2: a∈R  b∈R  c∈R
  shows
  a·(a·c) - b·(-b·c) = (a·a + b·b)·c
  a·(-b·c) + b·(a·c) = 0
proof -
  from A2 have T:
    b·c ∈ R  a·a ∈ R  b·b ∈ R
    b·(b·c) ∈ R  a·(b·c) ∈ R
    using  Ring_ZF_1_L4 by auto
  with A2 show
    a·(a·c) - b·(-b·c) = (a·a + b·b)·c
    using Ring_ZF_1_L7 Ring_ZF_1_L3 Ring_ZF_1_L11
      ring_oper_distr by simp
  from A2 T have
    a·(-b·c) + b·(a·c) = (-a·(b·c)) + b·a·c
    using Ring_ZF_1_L7 Ring_ZF_1_L11 by simp
  also from A1 A2 T have ... = 0
    using IsCommutative_def Ring_ZF_1_L11 Ring_ZF_1_L3
    by simp
  finally show a·(-b·c) + b·(a·c) = 0
    by simp
qed
```

Some rearrangements with four elements. Properties of abelian groups.

```
lemma (in ring0) Ring_ZF_2_L5:
  assumes a∈R  b∈R  c∈R  d∈R
  shows
  a - b - c - d = a - d - b - c
  a + b + c - d = a - d + b + c
  a + b - c - d = a - c + (b - d)
  a + b + c + d = a + c + (b + d)
  using assms Ring_ZF_1_L1(3) add_group.rearr_ab_gr_4_elemB
    add_group.rearr_ab_gr_4_elemA by auto
```

Two big rearranegements with six elements, useful for proving properties of complex addition and multiplication.

```
lemma (in ring0) Ring_ZF_2_L6:
  assumes A1: a∈R  b∈R  c∈R  d∈R  e∈R  f∈R
  shows
  a·(c·e - d·f) - b·(c·f + d·e) =
  (a·c - b·d)·e - (a·d + b·c)·f
  a·(c·f + d·e) + b·(c·e - d·f) =
  (a·c - b·d)·f + (a·d + b·c)·e
  a·(c+e) - b·(d+f) = a·c - b·d + (a·e - b·f)
  a·(d+f) + b·(c+e) = a·d + b·c + (a·f + b·e)
proof -
  from A1 have T:
    c·e ∈ R  d·f ∈ R  c·f ∈ R  d·e ∈ R
```

```
    a·c ∈ R  b·d ∈ R  a·d ∈ R  b·c ∈ R
    b·f ∈ R  a·e ∈ R  b·e ∈ R  a·f ∈ R
    a·c·e ∈ R  a·d·f ∈ R
    b·c·f ∈ R  b·d·e ∈ R
    b·c·e ∈ R  b·d·f ∈ R
    a·c·f ∈ R  a·d·e ∈ R
    a·c·e - a·d·f ∈ R
    a·c·e - b·d·e ∈ R
    a·c·f + a·d·e ∈ R
    a·c·f - b·d·f ∈ R
    a·c + a·e ∈ R
    a·d + a·f ∈ R
    using Ring_ZF_1_L4 by auto
  with A1 show a·(c·e - d·f) - b·(c·f + d·e) =
    (a·c - b·d)·e - (a·d + b·c)·f
    using Ring_ZF_1_L8 ring_oper_distr Ring_ZF_1_L11
      Ring_ZF_1_L10 Ring_ZF_2_L5 by simp
  from A1 T show
    a·(c·f + d·e) + b·(c·e - d·f) =
    (a·c - b·d)·f + (a·d + b·c)·e
    using Ring_ZF_1_L8 ring_oper_distr Ring_ZF_1_L11
    Ring_ZF_1_L10A Ring_ZF_2_L5 Ring_ZF_1_L10
    by simp
  from A1 T show
    a·(c+e) - b·(d+f) = a·c - b·d + (a·e - b·f)
    a·(d+f) + b·(c+e) = a·d + b·c + (a·f + b·e)
    using ring_oper_distr Ring_ZF_1_L10 Ring_ZF_2_L5
    by auto
qed

end
```

43 Binomial theorem

```
theory Ring_Binomial_ZF imports Monoid_ZF_1 Ring_ZF

begin
```

This theory aims at formalizing sufficient background to be able to state and prove the binomial theorem.

43.1 Sums of multiplicities of powers of ring elements and binomial theorem

The binomial theorem asserts that for any two elements of a commutative ring the n-th power of the sum $x + y$ can be written as a sum of certain multiplicities of terms $x^{n-k}y^k$, where $k \in 0..n$. In this section we setup the notation and prove basic properties of such multiplicities and powers of ring elements. We show the binomial theorem as an application.

The next locale (context) extends the `ring0` locale with notation for powers, multiplicities and sums and products of finite lists of ring elements.

```
locale ring3 = ring0 +

  fixes listprod (∏ _ 70)
  defines listprod_def [simp]: ∏s ≡ Fold(M,1,s)

  fixes pow
  defines pow_def [simp]: pow(n,x) ≡ ∏{⟨k,x⟩. k∈n}
```

A ring with addition forms a monoid, hence all propositions proven in the `monoid1` locale (defined in the `Monoid_ZF_1` theory) can be used in the `ring3` locale, applied to the additive operation.

```
sublocale ring0 < add_monoid: monoid1 R A ringa ringzero rlistsum rnat_mult

  using ringAssum
  unfolding IsAring_def IsAgroup_def monoid1_def monoid0_def
  by auto
```

A ring with multiplication forms a monoid, hence all propositions proven in the `monoid1` locale (defined in the `Monoid_ZF_1` theory) can be used in the `ring3` locale, applied to the multiplicative operation. (For some reason the sublocale below is not seen by Isabelle when we try to use it).

```
sublocale ring3 < mul_monoid: monoid1 R M ringm ringone listprod pow
  using ringAssum
  unfolding IsAring_def IsAgroup_def monoid1_def monoid0_def
  by auto
```

The assumptions of the `monoid1` context hold in the `ring0` context

```
lemma (in ring0) monoid0_valid_in_ring0: shows monoid1(R,A)
  using ringAssum
  unfolding IsAring_def IsAgroup_def monoid1_def monoid0_def
  by auto
```

$0 \cdot x = 0$ and $x^0 = 1$. It is a bit surprising that we do not need to assume that $x \in R$ (i.e. x is an element of the ring). These properties are really proven in the `Monoid_ZF_1` theory where there is no assumption that x is an element of the monoid.

```
lemma (in ring3) mult_pow_zero: shows 0·x = 0 and pow(0,x) = 1
  using monoid0_valid_in_ring0 monoid1.nat_mult_zero mul_monoid.nat_mult_zero
by simp_all
```

Natural multiple and power of a ring element is a ring element.

```
lemma (in ring3) mult_pow_type: assumes n∈nat x∈R
```

```
  shows n·x ∈ R and pow(n,x) ∈ R
  using assms monoid0_valid_in_ring0 monoid1.nat_mult_type mul_monoid.nat_mult_type

  by simp_all
```

The usual properties of multiples and powers: $(n+1)x = nx+x$ and $x^n+1 = x^nx$. These are just versions of `nat_mult_add_one` from `Monoid_ZF_1` writtent in the notation defined in the `ring3` locale.

```
lemma (in ring3) nat_mult_pow_add_one: assumes  n∈nat x∈R
  shows (n #+ 1)·x = (n·x) + x and pow(n #+ 1,x) = pow(n,x)·x
  using assms monoid0_valid_in_ring0 monoid1.nat_mult_add_one mul_monoid.nat_mult_add_one

  by simp_all
```

Associativity for the multiplication by natural number and the ring multiplication:

```
lemma (in ring3) nat_mult_assoc: assumes n∈nat x∈R y∈R
  shows n·x·y = n·(x·y)
proof -
  from assms(1,3) have n∈nat and 0·x·y = 0·(x·y)
    using mult_pow_zero(1) Ring_ZF_1_L6 by simp_all
  moreover from assms(2,3) have ∀k∈nat.
    k·x·y = k·(x·y) ⟶ (k #+ 1)·x·y = (k #+ 1)·(x·y)
    using nat_mult_pow_add_one(1) mult_pow_type ring_oper_distr(2) Ring_ZF_1_L4(3)
      by simp
  ultimately show thesis by (rule ind_on_nat1)
qed
```

Addition of natural numbers is distributive with respect to natural multiple. This is essentially lemma `nat_mult_add` from `Monoid_ZF_1.thy`, just transferred to the `ring3` locale.

```
lemma (in ring3) nat_add_mult_distrib: assumes n∈nat m∈nat x∈R
  shows (n #+ m)·x = n·x + m·x
  using assms monoid0_valid_in_ring0 monoid1.nat_mult_add by simp
```

Associativity for the multiplication by natural number and the ring multiplication extended to three elements of the ring:

```
lemma (in ring3) nat_mult_assoc1: assumes n∈nat x∈R y∈R z∈R
  shows n·x·y·z = n·(x·y·z)
  using assms Ring_ZF_1_L4(3) nat_mult_assoc by simp
```

When we multiply an expression whose value belongs to a ring by a ring element and we get an expression whose value belongs to a ring.

```
lemma (in ring3) mult_elem_ring_type:
  assumes n∈nat x∈R and ∀k∈n. q(k) ∈ R
  shows ∀k∈n. q(k)·x ∈ R and (∑{⟨k,q(k)·x⟩. k∈n}) ∈ R
  using assms Ring_ZF_1_L4(3) monoid0_valid_in_ring0 monoid1.sum_in_mono
by simp_all
```

The sum of expressions whose values belong to a ring is an expression whose value belongs to a ring.

```
lemma (in ring3) sum_expr_ring_type:
  assumes n∈nat ∀k∈n. q(k) ∈ R ∀k∈n. p(k) ∈ R
  shows ∀k∈n. q(k)+p(k) ∈ R and (∑{⟨k,q(k)+p(k)⟩. k∈n}) ∈ R
  using assms Ring_ZF_1_L4(1) monoid0_valid_in_ring0 monoid1.sum_in_mono
by simp_all
```

Combining `mult_elem_ring_type` and `sum_expr_ring_type` we obtain that a (kind of) linear combination of expressions whose values belong to a ring belongs to the ring.

```
lemma (in ring3) lin_comb_expr_ring_type:
  assumes n∈nat x∈R  y∈R ∀k∈n. q(k) ∈ R ∀k∈n. p(k) ∈ R
  shows ∀k∈n. q(k)·x+p(k)·y ∈ R and
    (∑{⟨k,q(k)·x+p(k)·y⟩. k∈n}) ∈ R
proof -
  from assms have ∀k∈n. q(k)·x ∈ R and ∀k∈n. p(k)·y ∈ R
    using mult_elem_ring_type(1) by simp_all
  with assms(1) show ∀k∈n. q(k)·x+p(k)·y ∈ R
    using sum_expr_ring_type(1) by simp
  with assms(1) show (∑{⟨k,q(k)·x+p(k)·y⟩. k∈n}) ∈ R
    using monoid0_valid_in_ring0 monoid1.sum_in_mono by simp
qed
```

A `ring3` version of `seq_sum_pull_one_elem` from `Monoid_ZF_1`:

```
lemma (in ring3) rng_seq_sum_pull_one_elem:
  assumes j ∈ nat ∀k∈j #+ 1. q(k) ∈ R
  shows
    (∑{⟨k,q(k)⟩. k∈j #+ 1}) = q(0)+(∑{⟨k,q(k #+ 1)⟩. k∈j})
    (∑{⟨k,q(k)⟩. k∈j #+ 1}) = (∑{⟨k,q(k)⟩. k∈j})+ q(j)
  using assms monoid0_valid_in_ring0 monoid1.seq_sum_pull_one_elem by
simp_all
```

Distributive laws for finite sums in a ring: $(\sum_{k=0}^{n-1} q(k)) \cdot x = \sum_{k=0}^{n-1} q(k) \cdot x$ and $x \cdot (\sum_{k=0}^{n-1} q(k)) = \sum_{k=0}^{n-1} x \cdot q(k)$.

```
theorem (in ring3) fin_sum_distrib:
  assumes x∈R  n∈nat ∀k∈n. q(k) ∈ R
  shows
    (∑{⟨k,q(k)⟩. k∈n})·x = ∑{⟨k,q(k)·x⟩. k∈n}
    x·(∑{⟨k,q(k)⟩. k∈n}) = ∑{⟨k,x·q(k)⟩. k∈n}
proof -
  from assms(1,2) have n∈nat and
    (∑{⟨k,q(k)⟩. k∈0})·x = ∑{⟨k,q(k)·x⟩. k∈0}
    using monoid0_valid_in_ring0 monoid1.sum_empty Ring_ZF_1_L6(1) by
simp_all
  moreover have
    ∀j∈n. (∑{⟨k,q(k)⟩. k∈j})·x = (∑{⟨k,q(k)·x⟩. k∈j})
    ⟶ (∑{⟨k,q(k)⟩. k∈j #+ 1})·x = ∑{⟨k,q(k)·x⟩. k∈j #+ 1}
```

```
  proof -
    { fix j assume j∈n and
        I: (∑{⟨k,q(k)⟩. k∈j})·x = (∑{⟨k,q(k)·x⟩. k∈j})
      from assms(2) ‹j∈n› have j∈nat using elem_nat_is_nat(2)
        by simp
      moreover from assms(2,3) ‹j∈n› have II: ∀k∈j #+ 1. q(k) ∈ R
        using mem_add_one_subset by blast
      ultimately have
        (∑{⟨k,q(k)⟩. k∈j #+ 1}) =  (∑{⟨k,q(k)⟩. k∈j}) + q(j)
        using monoid0_valid_in_ring0 monoid1.seq_sum_pull_one_elem(2)
by simp
      hence
        (∑{⟨k,q(k)⟩. k∈j #+ 1})·x = ((∑{⟨k,q(k)⟩. k∈j}) + q(j))·x
        by simp
      moreover from assms(1) ‹j∈nat› II have
        (∑{⟨k,q(k)⟩. k∈j}) ∈ R q(j) ∈ R and x∈R
        using monoid0_valid_in_ring0 monoid1.sum_in_mono by simp_all
      ultimately have
        (∑{⟨k,q(k)⟩. k∈j #+ 1})·x = (∑{⟨k,q(k)⟩. k∈j})·x + q(j)·x
        using ring_oper_distr(2) by simp
      with I have
        (∑{⟨k,q(k)⟩. k∈j #+ 1})·x = (∑{⟨k,q(k)·x⟩. k∈j})  + q(j)·x
        by simp
      moreover
      from assms(1) II have ∀k∈j #+ 1. q(k)·x ∈ R
        using Ring_ZF_1_L4(3) by simp
      with ‹j∈nat› have
        (∑{⟨k,q(k)·x⟩. k∈j #+ 1}) = (∑{⟨k,q(k)·x⟩. k∈j}) + q(j)·x
        using monoid0_valid_in_ring0 monoid1.seq_sum_pull_one_elem(2)
by simp
      ultimately have
        (∑{⟨k,q(k)⟩. k∈j #+ 1})·x = (∑{⟨k,q(k)·x⟩. k∈j #+ 1})
        by simp
    } thus thesis by simp
  qed
  ultimately show (∑{⟨k,q(k)⟩. k∈n})·x = ∑{⟨k,q(k)·x⟩. k∈n}
    by (rule fin_nat_ind1)
  from assms(1,2) have n∈nat and
    x·(∑{⟨k,q(k)⟩. k∈0}) = ∑{⟨k,x·q(k)⟩. k∈0}
    using monoid0_valid_in_ring0 monoid1.sum_empty Ring_ZF_1_L6(2) by
simp_all
  moreover have
    ∀j∈n. x·(∑{⟨k,q(k)⟩. k∈j}) = (∑{⟨k,x·q(k)⟩. k∈j})
    ⟶ x·(∑{⟨k,q(k)⟩. k∈j #+ 1}) = ∑{⟨k,x·q(k)⟩. k∈j #+ 1}
  proof -
    { fix j assume j∈n and
        I: x·(∑{⟨k,q(k)⟩. k∈j}) = (∑{⟨k,x·q(k)⟩. k∈j})
      from assms(2) ‹j∈n› have j∈nat using elem_nat_is_nat(2)
        by simp
```

```
      moreover from assms(2,3) ‹j∈n› have II: ∀k∈j #+ 1. q(k) ∈ R
        using mem_add_one_subset by blast
      ultimately have
        (∑{⟨k,q(k)⟩. k∈j #+ 1}) =  (∑{⟨k,q(k)⟩. k∈j}) + q(j)
        using monoid0_valid_in_ring0 monoid1.seq_sum_pull_one_elem(2)
by simp
      hence
        x·(∑{⟨k,q(k)⟩. k∈j #+ 1}) = x·((∑{⟨k,q(k)⟩. k∈j}) + q(j))
        by simp
      moreover from assms(1) ‹j∈nat› II have
        (∑{⟨k,q(k)⟩. k∈j}) ∈ R q(j) ∈ R and x∈R
        using monoid0_valid_in_ring0 monoid1.sum_in_mono by simp_all
       ultimately have
        x·(∑{⟨k,q(k)⟩. k∈j #+ 1}) = x·(∑{⟨k,q(k)⟩. k∈j}) + x·q(j)
        using ring_oper_distr(1) by simp
       with I have
        x·(∑{⟨k,q(k)⟩. k∈j #+ 1}) =  (∑{⟨k,x·q(k)⟩. k∈j})  + x·q(j)
        by simp
      moreover
      from assms(1) II have ∀k∈j #+ 1. x·q(k) ∈ R
        using Ring_ZF_1_L4(3) by simp
      with ‹j∈nat› have
        (∑{⟨k,x·q(k)⟩. k∈j #+ 1}) = (∑{⟨k,x·q(k)⟩. k∈j}) + x·q(j)
        using monoid0_valid_in_ring0 monoid1.seq_sum_pull_one_elem(2)
by simp
      ultimately have
        x·(∑{⟨k,q(k)⟩. k∈j #+ 1}) = (∑{⟨k,x·q(k)⟩. k∈j #+ 1})
        by simp
    } thus thesis by simp
  qed
  ultimately show  x·(∑{⟨k,q(k)⟩. k∈n}) = ∑{⟨k,x·q(k)⟩. k∈n}
    by (rule fin_nat_ind1)
qed
```

In rings we have $\sum_{k=0}^{n-1} q(k)+p(k) = (\sum_{k=0}^{n-1} p(k))+(\sum_{k=0}^{n-1} q(k))$. This is the same as theorem `sum_comm_distrib` in `Monoid_ZF_1.thy`, except that we do not need the assumption about commutativity of the operation as addition in rings is always commutative.

```
lemma (in ring3) sum_ring_distrib:
  assumes n∈nat and  ∀k∈n. p(k) ∈ R ∀k∈n. q(k) ∈ R
  shows
    (∑{⟨k,p(k)+q(k)⟩. k∈n}) = (∑{⟨k,p(k)⟩. k∈n}) + (∑{⟨k,q(k)⟩. k∈n})
  using assms Ring_ZF_1_L1(3) monoid0_valid_in_ring0 monoid1.sum_comm_distrib
by simp
```

To shorten the notation in the proof of the binomial theorem we give a name to the binomial term $\binom{n}{k}x^{n-k}y^k$.

```
definition (in ring3) BT where
  BT(n,k,x,y) ≡ Binom(n,k)·pow(n #- k,x)·pow(k,y)
```

If n, k are natural numbers and x, y are ring elements then the binomial term is an element of the ring.

```
lemma (in ring3) bt_type: assumes n∈nat k∈nat x∈R y∈R
  shows BT(n,k,x,y) ∈ R
  using assms mult_pow_type binom_in_nat Ring_ZF_1_L4(3)
  unfolding BT_def by simp
```

The binomial term is 1 when the $n = 0$ and $k = 0$. Somehow we do not need the assumption that x, y are ring elements.

```
lemma (in ring3) bt_at_zero: shows BT(0,0,x,y) = 1
  using binom_zero_zero mult_pow_zero(2) monoid0_valid_in_ring0 monoid1.nat_mult_one

        Ring_ZF_1_L2(2) Ring_ZF_1_L3(5)
  unfolding BT_def by simp
```

The binomial term is x^n when $k = 0$.

```
lemma (in ring3) bt_at_zero1: assumes n∈nat x∈R
  shows BT(n,0,x,y) = pow(n,x)
  unfolding BT_def using assms mult_pow_zero(2) binom_left_boundary
    mult_pow_type(2) monoid0_valid_in_ring0 monoid1.nat_mult_one Ring_ZF_1_L3(5)

    by simp
```

When $k = 0$ multiplying the binomial term by x is the same as adding one to n.

```
lemma (in ring3) bt_at_zero2: assumes n∈nat x∈R
  shows BT(n,0,x,y)·x = BT(n #+ 1,0,x,y)
  using assms bt_at_zero1 nat_mult_pow_add_one(2) by simp
```

The binomial term is y^n when $k = n$.

```
lemma (in ring3) bt_at_right: assumes n∈nat y∈R
  shows BT(n,n,x,y) = pow(n,y)
  unfolding BT_def using assms binom_right_boundary mult_pow_zero(2)
    monoid0_valid_in_ring0 monoid1.nat_mult_one Ring_ZF_1_L2(2) mult_pow_type(2)
Ring_ZF_1_L3(6)
  by simp
```

When $k = n$ multiplying the binomial term by x is the same as adding one to n.

```
lemma (in ring3) bt_at_right1: assumes n∈nat y∈R
  shows BT(n,n,x,y)·y = BT(n #+ 1,n #+ 1,x,y)
  using assms bt_at_right nat_mult_pow_add_one(2) by simp
```

A key identity for binomial terms needed for the proof of the binomial theorem:

```
lemma (in ring3) bt_rec_identity:
  assumes M {is commutative on} R j∈nat k∈j x∈R y∈R
```

```
  shows
    BT(j,k #+ 1,x,y)·x + BT(j,k,x,y)·y = BT(j #+ 1,k #+ 1,x,y)
proof -
  from assms(2,3,4) have k∈nat Binom(j,k #+ 1) ∈ nat
    and Binom(j,k) ∈ nat Binom(j #+ 1,k #+ 1) ∈ nat
    and I: pow(j #- (k #+ 1),x) ∈ R and II: pow(j #- k,x) ∈ R
    using elem_nat_is_nat(2) binom_in_nat mult_pow_type(2)
    by simp_all
  with assms(5) have III: pow(k #+ 1,y) ∈ R
    using mult_pow_type(2) by blast
  let L = BT(j,k #+ 1,x,y)·x + BT(j,k,x,y)·y
  let p = pow(j #- k,x)·pow(k #+ 1,y)
  from assms(2,4) ‹k∈nat› II III have p ∈ R
    using mult_pow_type(2) Ring_ZF_1_L4(3) by simp
  from assms(2,3,4,5) have BT(j,k,x,y)·y = Binom(j,k)·p
    using elem_nat_is_nat(2) binom_in_nat mult_pow_type(2)
      nat_mult_assoc1 Ring_ZF_1_L11(2) nat_mult_pow_add_one(2)
    unfolding BT_def by simp
  moreover have BT(j,k #+ 1,x,y)·x = Binom(j,k #+ 1)·p
  proof -
    from assms(1,4) ‹Binom(j,k #+ 1) ∈ nat› I III have
      Binom(j,k #+ 1)·pow(j #- (k #+ 1),x)·pow(k #+ 1,y)·x =
        Binom(j,k #+ 1)·(pow(j #- (k #+ 1) #+ 1,x)·pow(k #+ 1,y))
      using nat_mult_assoc1 Ring_ZF_2_L4(2) nat_mult_pow_add_one(2)
      by simp
    with assms(2,3) have
      Binom(j,k #+ 1)·pow(j #- (k #+ 1),x)·pow(k #+ 1,y)·x =
      Binom(j,k #+ 1)·(pow(j #- (k #+ 1) #+ 1,x)·pow(k #+ 1,y))
      using nat_subtr_simpl0 by blast
    moreover from assms(2,3) have
      pow(j #- (k #+ 1) #+ 1,x) = pow(j #- k,x)
      using nat_subtr_simpl0 by simp
    ultimately show thesis unfolding BT_def by simp
  qed
  ultimately have L = Binom(j,k #+ 1)·p + Binom(j,k)·p
    by simp
  with assms(2,3) ‹Binom(j,k #+ 1) ∈ nat› ‹Binom(j,k) ∈ nat› ‹p ∈ R›
  have L = Binom(j #+ 1,k #+ 1)·p
    using nat_add_mult_distrib binom_prop2 succ_ineq1(3) by simp
  with assms(2,3) ‹Binom(j #+ 1,k #+ 1) ∈ nat› II III
  show thesis using nat_mult_assoc nat_subtr_simpl0
    unfolding BT_def by simp
qed
```

The binomial theorem: if x, y are elements of a commutative ring, $n \in \mathbb{N}$ then $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$.

```
theorem (in ring3) binomial_theorem:
  assumes M {is commutative on} R n∈nat x∈R y∈R
  shows
```

```
    pow(n,x+y) = ∑{⟨k,Binom(n,k)·pow(n #- k,x) · pow(k,y)⟩. k∈n #+ 1}
proof -
  from assms(2) have n∈nat by simp
  moreover have pow(0,x+y) = ∑{⟨k,BT(0,k,x,y)⟩. k∈0 #+ 1}
  proof -
    from assms(3,4) have (∑{⟨k,BT(0,k,x,y)⟩. k∈0 #+ 1}) = 1
      using bt_at_zero Ring_ZF_1_L2(2) monoid0_valid_in_ring0 monoid1.seq_sum_singleton
      by simp
    then show thesis using mult_pow_zero(2) by simp
  qed
  moreover have ∀j∈nat.
    pow(j,x+y) = (∑{⟨k,BT(j,k,x,y)⟩. k∈j #+ 1})  ⟶
    pow(j #+ 1,x+y) = (∑{⟨k,BT(j #+ 1,k,x,y)⟩. k∈j #+ 1 #+ 1})
  proof -
    { fix j
      let s0 = ∑{⟨k,BT(j,k,x,y)⟩. k∈j #+ 1}
      let s1 = ∑{⟨k,BT(j,k,x,y)·x⟩. k∈j #+ 1}
      let s2 = ∑{⟨k,BT(j,k,x,y)·y⟩. k∈j #+ 1}
      let s3 = ∑{⟨k,BT(j,k #+ 1,x,y)·x⟩. k∈j}
      let s4 = ∑{⟨k,BT(j,k,x,y)·y⟩. k∈j}
      let s5 = ∑{⟨k,BT(j,k #+ 1,x,y)·x + BT(j,k,x,y)·y⟩. k∈j}
      let s6 = ∑{⟨k,BT(j #+ 1,k #+ 1,x,y)⟩. k∈j}
      let s7 = ∑{⟨k,BT(j #+ 1,k,x,y)⟩. k∈j #+ 1}
      let s8 = ∑{⟨k,BT(j #+ 1,k,x,y)⟩. k∈j #+ 1 #+ 1}
      assume j∈nat and pow(j,x+y) = s0
      then have j #+ 1 ∈ nat and j #+ 1 #+ 1 ∈ nat by simp_all
      have
        I:  ∀k∈j #+ 1. BT(j,k,x,y) ∈ R and
        II: ∀k∈j #+ 1. BT(j,k,x,y)·x ∈ R and
        III: ∀k∈j #+ 1. BT(j,k,x,y)·y ∈ R and
        IV:  ∀k∈j. BT(j,k #+ 1,x,y)·x ∈ R and
        V:   ∀k∈j. BT(j,k,x,y)·y ∈ R and
        VI:  ∀k∈j #+ 1. BT(j #+ 1,k,x,y) ∈ R and
        VII: ∀k∈j #+ 1 #+ 1. BT(j #+ 1,k,x,y) ∈ R
      proof -
        from assms(3,4) ‹j∈nat› show ∀k∈j #+ 1. BT(j,k,x,y) ∈ R
          using elem_nat_is_nat(2) bt_type by blast
        with assms(3,4) ‹j∈nat› show
          ∀k∈j #+ 1. BT(j,k,x,y)·x ∈ R and
          ∀k∈j #+ 1. BT(j,k,x,y)·y ∈ R and
          ∀k∈j. BT(j,k,x,y)·y ∈ R
          using Ring_ZF_1_L4(3) by simp_all
        from assms(3,4) ‹j∈nat› have ∀k∈j. BT(j,k #+ 1,x,y) ∈ R
          using elem_nat_is_nat(2) bt_type by simp
        with ‹j∈nat› assms(3) show ∀k∈j. BT(j,k #+ 1,x,y)·x ∈ R
          using mult_elem_ring_type(1) by simp
        from assms(3,4) ‹j #+ 1 ∈ nat› show
          ∀k∈j #+ 1. BT(j #+ 1,k,x,y) ∈ R
          using elem_nat_is_nat(2) bt_type by blast
```

```
      from assms(3,4) ‹j #+ 1 #+ 1 ∈ nat› show
        ∀k∈j #+ 1 #+ 1. BT(j #+ 1,k,x,y) ∈ R
        using elem_nat_is_nat(2) bt_type by blast
    qed
    have pow(j #+ 1,x+y) = s0·x + s0·y
    proof -
      from assms(3,4) ‹j∈nat› have
        pow(j #+ 1,x+y) = pow(j,x+y)·x + pow(j,x+y)·y
        using Ring_ZF_1_L4(1) mult_pow_type nat_mult_pow_add_one(2)

          ring_oper_distr(1) by simp
      with ‹pow(j,x+y) = s0› show  thesis by simp
    qed
    also have s0·x + s0·y = s1 + s2
    proof -
      from assms(3) ‹j #+ 1 ∈ nat› I have s0·x = s1
        by (rule fin_sum_distrib)
      moreover from assms(4) ‹j #+ 1 ∈ nat› I
      have s0·y = s2 by (rule fin_sum_distrib)
      ultimately show thesis by simp
    qed
    also have s1 + s2 =
      (BT(j,0,x,y)·x + s3) + (s4 + BT(j,j,x,y)·y)
    proof -
      from ‹j∈nat› II have s1 = BT(j,0,x,y)·x + s3
        using rng_seq_sum_pull_one_elem(1) by simp
      moreover from ‹j∈nat› III have s2 = s4 + BT(j,j,x,y)·y
        using rng_seq_sum_pull_one_elem(2) by simp
      ultimately show thesis by simp
    qed
    also from assms(3,4) IV V ‹j∈nat› have
      (BT(j,0,x,y)·x + s3) + (s4 + BT(j,j,x,y)·y) =
      BT(j,0,x,y)·x + (s3 + s4) + BT(j,j,x,y)·y
      using bt_type Ring_ZF_1_L4(3) monoid0_valid_in_ring0 monoid1.sum_in_mono
Ring_ZF_2_L2(3)
      by simp
    also have BT(j,0,x,y)·x + (s3 + s4) + BT(j,j,x,y)·y =
      BT(j,0,x,y)·x + s5 + BT(j,j,x,y)·y
    proof -
      from ‹j∈nat› IV V have s3 + s4 = s5
        using sum_ring_distrib by simp
      thus thesis by simp
    qed
    also from assms(1,3,4) ‹j∈nat› have
      BT(j,0,x,y)·x + s5 + BT(j,j,x,y)·y =
      BT(j,0,x,y)·x + s6 + BT(j,j,x,y)·y
      using bt_rec_identity by simp
    also have BT(j,0,x,y)·x + s6 + BT(j,j,x,y)·y =
      s7 + BT(j,j,x,y)·y
```

```
      proof -
        from ‹j∈nat› VI have s₇ = BT(j #+ 1,0,x,y) + s₆
          by (rule rng_seq_sum_pull_one_elem)
        with assms(3) ‹j∈nat› show thesis
          using bt_at_zero2 by simp
      qed
      also have s₇ + BT(j,j,x,y)·y = s₈
      proof -
        from ‹j #+ 1 ∈ nat› VII have
          s₈ = s₇ + BT(j #+ 1,j #+ 1,x,y)
          by (rule rng_seq_sum_pull_one_elem)
        with assms(4) ‹j∈nat› show thesis
          using bt_at_right1 by simp
      qed
      finally have pow(j #+ 1,x+y) =  s₈ by simp
    } thus thesis by simp
  qed
  ultimately have pow(n,x+y) = ∑{⟨k,BT(n,k,x,y)⟩. k∈n #+ 1}
    by (rule ind_on_nat1)
  then show thesis unfolding BT_def by simp
qed

end
```

44 More on rings

```
theory Ring_ZF_1 imports Ring_ZF Group_ZF_3

begin
```

This theory is devoted to the part of ring theory specific the construction of real numbers in the `Real_ZF_x` series of theories. The goal is to show that classes of almost homomorphisms form a ring.

44.1 The ring of classes of almost homomorphisms

Almost homomorphisms do not form a ring as the regular homomorphisms do because the lifted group operation is not distributive with respect to composition – we have $s \circ (r \cdot q) \neq s \circ r \cdot s \circ q$ in general. However, we do have $s \circ (r \cdot q) \approx s \circ r \cdot s \circ q$ in the sense of the equivalence relation defined by the group of finite range functions (that is a normal subgroup of almost homomorphisms, if the group is abelian). This allows to define a natural ring structure on the classes of almost homomorphisms.

The next lemma provides a formula useful for proving that two sides of the distributive law equation for almost homomorphisms are almost equal.

```
lemma (in group1) Ring_ZF_1_1_L1:
```

```
  assumes A1: s∈AH r∈AH q∈AH and A2: n∈G
  shows
  ((s∘(r·q))(n))·(((s∘r)·(s∘q))(n))⁻¹= δ(s,⟨ r(n),q(n)⟩)
  ((r·q)∘s)(n) = ((r∘s)·(q∘s))(n)
proof -
  from groupAssum isAbelian A1 have T1:
    r·q ∈ AH s∘r ∈ AH  s∘q ∈ AH (s∘r)·(s∘q) ∈ AH
    r∘s ∈ AH q∘s ∈ AH  (r∘s)·(q∘s) ∈ AH
    using Group_ZF_3_2_L15 Group_ZF_3_4_T1 by auto
  from A1 A2 have T2: r(n) ∈ G q(n) ∈ G s(n) ∈ G
    s(r(n)) ∈ G s(q(n)) ∈ G δ(s,⟨ r(n),q(n)⟩) ∈ G
    s(r(n))·s(q(n)) ∈ G r(s(n)) ∈ G q(s(n)) ∈ G
    r(s(n))·q(s(n)) ∈ G
    using AlmostHoms_def apply_funtype Group_ZF_3_2_L4B
    group0_2_L1 monoid0.group0_1_L1 by auto
  with T1 A1 A2 isAbelian show
    ((s∘(r·q))(n))·(((s∘r)·(s∘q))(n))⁻¹= δ(s,⟨ r(n),q(n)⟩)
    ((r·q)∘s)(n) = ((r∘s)·(q∘s))(n)
    using Group_ZF_3_2_L12 Group_ZF_3_4_L2 Group_ZF_3_4_L1 group0_4_L6A
    by auto
qed
```

The sides of the distributive law equations for almost homomorphisms are almost equal.

```
lemma (in group1) Ring_ZF_1_1_L2:
  assumes A1: s∈AH r∈AH q∈AH
  shows
  s∘(r·q) ≅ (s∘r)·(s∘q)
  (r·q)∘s = (r∘s)·(q∘s)
proof -
  from A1 have ∀n∈G. ⟨ r(n),q(n)⟩ ∈ G×G
    using AlmostHoms_def apply_funtype by auto
  moreover from A1 have {δ(s,x). x ∈ G×G} ∈ Fin(G)
    using AlmostHoms_def by simp
  ultimately have {δ(s,⟨ r(n),q(n)⟩). n∈G} ∈ Fin(G)
    by (rule Finite1_L6B)
  with A1 have
    {((s∘(r·q))(n))·(((s∘r)·(s∘q))(n))⁻¹. n ∈ G} ∈ Fin(G)
    using Ring_ZF_1_1_L1 by simp
  moreover from groupAssum isAbelian A1 A1 have
    s∘(r·q) ∈ AH (s∘r)·(s∘q) ∈ AH
    using Group_ZF_3_2_L15 Group_ZF_3_4_T1 by auto
  ultimately show s∘(r·q) ≅ (s∘r)·(s∘q)
    using Group_ZF_3_4_L12 by simp
  from groupAssum isAbelian A1 have
    (r·q)∘s : G→G (r∘s)·(q∘s) : G→G
    using Group_ZF_3_2_L15 Group_ZF_3_4_T1 AlmostHoms_def
    by auto
  moreover from A1 have
```

```
    ∀n∈G. ((r·q)∘s)(n) = ((r∘s)·(q∘s))(n)
    using Ring_ZF_1_1_L1 by simp
  ultimately show (r·q)∘s = (r∘s)·(q∘s)
    using fun_extension_iff by simp
qed
```

The essential condition to show the distributivity for the operations defined on classes of almost homomorphisms.

```
lemma (in group1) Ring_ZF_1_1_L3:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  and A2: a ∈ AH//R b ∈ AH//R c ∈ AH//R
  and A3: A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows M⟨a,A⟨ b,c⟩⟩ = A⟨M⟨ a,b⟩,M⟨ a,c⟩⟩ ∧
  M⟨A⟨ b,c⟩,a⟩ = A⟨M⟨ b,a⟩,M⟨ c,a⟩⟩
proof
  from A2 obtain s q r where D1: s∈AH r∈AH q∈AH
    a = R{s} b = R{q} c = R{r}
    using quotient_def by auto
  from A1 have T1:equiv(AH,R)
      using Group_ZF_3_3_L3 by simp
  with A1 A3 D1 groupAssum isAbelian have
    M⟨  a,A⟨ b,c⟩ ⟩ = R{s∘(q·r)}
    using Group_ZF_3_3_L4 EquivClass_1_L10
    Group_ZF_3_2_L15 Group_ZF_3_4_L13A by simp
  also have R{s∘(q·r)} = R{(s∘q)·(s∘r)}
  proof -
    from T1 D1 have equiv(AH,R) s∘(q·r) ≅ (s∘q)·(s∘r)
      using Ring_ZF_1_1_L2 by auto
    with A1 show thesis using equiv_class_eq by simp
  qed
  also from A1 T1 D1 A3 have
    R{(s∘q)·(s∘r)} = A⟨M⟨ a,b⟩,M⟨ a,c⟩⟩
    using Group_ZF_3_3_L4 Group_ZF_3_4_T1 EquivClass_1_L10
    Group_ZF_3_3_L3 Group_ZF_3_4_L13A EquivClass_1_L10 Group_ZF_3_4_T1
    by simp
  finally show M⟨a,A⟨ b,c⟩⟩ = A⟨M⟨ a,b⟩,M⟨ a,c⟩⟩ by simp
  from A1 A3 T1 D1 groupAssum isAbelian show
    M⟨A⟨ b,c⟩,a⟩ = A⟨M⟨ b,a⟩,M⟨ c,a⟩⟩
    using Group_ZF_3_3_L4 EquivClass_1_L10 Group_ZF_3_4_L13A
      Group_ZF_3_2_L15 Ring_ZF_1_1_L2 Group_ZF_3_4_T1 by simp
qed
```

The projection of the first group operation on almost homomorphisms is distributive with respect to the second group operation.

```
lemma (in group1) Ring_ZF_1_1_L4:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  and A2: A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows IsDistributive(AH//R,A,M)
proof -
```

```
  from A1 A2 have ∀a∈(AH//R).∀b∈(AH//R).∀c∈(AH//R).
  M⟨a,A⟨ b,c⟩⟩ = A⟨M⟨ a,b⟩, M⟨ a,c⟩⟩ ∧
  M⟨A⟨ b,c⟩, a⟩ = A⟨M⟨ b,a⟩,M⟨ c,a⟩⟩
    using Ring_ZF_1_1_L3 by simp
  then show thesis using IsDistributive_def by simp
qed
```

The classes of almost homomorphisms form a ring.

```
theorem (in group1) Ring_ZF_1_1_T1:
  assumes R = QuotientGroupRel(AH,Op1,FR)
  and A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows IsAring(AH//R,A,M)
  using assms QuotientGroupOp_def Group_ZF_3_3_T1 Group_ZF_3_4_T2
    Ring_ZF_1_1_L4 IsAring_def by simp

end
```

45 Ordered rings

```
theory OrderedRing_ZF imports Ring_ZF OrderedGroup_ZF_1

begin
```

In this theory file we consider ordered rings.

45.1 Definition and notation

This section defines ordered rings and sets up appriopriate notation.

We define ordered ring as a commutative ring with linear order that is preserved by translations and such that the set of nonnegative elements is closed under multiplication. Note that this definition does not guarantee that there are no zero divisors in the ring.

```
definition
  IsAnOrdRing(R,A,M,r) ≡
  ( IsAring(R,A,M) ∧ (M {is commutative on} R) ∧
  r⊆R×R ∧ IsLinOrder(R,r) ∧
  (∀a b. ∀ c∈R. ⟨ a,b⟩ ∈ r ⟶ ⟨A⟨ a,c⟩,A⟨ b,c⟩⟩ ∈ r) ∧
  (Nonnegative(R,A,r) {is closed under} M))
```

The next context (locale) defines notation used for ordered rings. We do that by extending the notation defined in the `ring0` locale and adding some assumptions to make sure we are talking about ordered rings in this context.

```
locale ring1 = ring0 +

  assumes mult_commut: M {is commutative on} R
```

```
  fixes r

  assumes ordincl: r ⊆ R×R

  assumes linord: IsLinOrder(R,r)

  fixes lesseq (infix ≤ 68)
  defines lesseq_def [simp]: a ≤ b ≡ ⟨ a,b⟩ ∈ r

  fixes sless (infix < 68)
  defines sless_def [simp]: a < b ≡ a≤b ∧ a≠b

  assumes ordgroup: ∀a b. ∀ c∈R. a≤b ⟶ a+c ≤ b+c

  assumes pos_mult_closed: Nonnegative(R,A,r) {is closed under} M

  fixes abs (| _ |)
  defines abs_def [simp]: |a| ≡ AbsoluteValue(R,A,r)(a)

  fixes positiveset (R+)
  defines positiveset_def [simp]: R+ ≡ PositiveSet(R,A,r)
```

The next lemma assures us that we are talking about ordered rings in the `ring1` context.

```
lemma (in ring1) OrdRing_ZF_1_L1: shows IsAnOrdRing(R,A,M,r)
  using ring0_def ringAssum mult_commut ordincl linord ordgroup
    pos_mult_closed IsAnOrdRing_def by simp
```

We can use theorems proven in the `ring1` context whenever we talk about an ordered ring.

```
lemma OrdRing_ZF_1_L2: assumes IsAnOrdRing(R,A,M,r)
  shows ring1(R,A,M,r)
  using assms IsAnOrdRing_def ring1_axioms.intro ring0_def ring1_def
  by simp
```

In the `ring1` context $a \leq b$ implies that a, b are elements of the ring.

```
lemma (in ring1) OrdRing_ZF_1_L3: assumes a≤b
  shows a∈R  b∈R
  using assms ordincl by auto
```

In the `ring1` context $a < b$ implies that a, b are elements of the ring.

```
lemma (in  ring1) ord_ring_less_members: assumes a<b
  shows a∈R  b∈R
  using assms OrdRing_ZF_1_L3 by auto
```

Ordered ring is an ordered group, hence we can use theorems proven in the `group3` context.

```
lemma (in  ring1) OrdRing_ZF_1_L4: shows
```

```
  IsAnOrdGroup(R,A,r)
  r {is total on} R
  A {is commutative on} R
  group3(R,A,r)
proof -
  { fix a b g assume A1: g∈R and A2: a≤b
    with ordgroup have a+g ≤ b+g
      by simp
    moreover from ringAssum A1 A2 have
      a+g = g+a  b+g = g+b
      using OrdRing_ZF_1_L3 IsAring_def IsCommutative_def by auto
    ultimately have
      a+g ≤ b+g  g+a ≤ g+b
      by auto
  } hence
    ∀g∈R. ∀a b. a≤b ⟶ a+g ≤ b+g ∧ g+a ≤ g+b
    by simp
  with ringAssum ordincl linord show
    IsAnOrdGroup(R,A,r)
    group3(R,A,r)
    r {is total on} R
    A {is commutative on} R
    using IsAring_def Order_ZF_1_L2 IsAnOrdGroup_def group3_def IsLinOrder_def
    by auto
qed
```

We can express that x is positive by stating that $0 < x$ or by writing that x is an element R_+.

```
lemma (in  ring1) element_pos: shows a∈R₊ ⟷ 0<a
  using OrdRing_ZF_1_L4(4) group3.OrderedGroup_ZF_1_L2A by auto
```

The order relation in rings is transitive.

```
lemma (in ring1) ring_ord_transitive: assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    group3(R,A,r)  ⟨a,b⟩ ∈ r   ⟨b,c⟩ ∈ r
    using OrdRing_ZF_1_L4 by auto
  then have ⟨a,c⟩ ∈ r by (rule group3.Group_order_transitive)
  then show  a≤c by simp
qed
```

Transitivity for the strict order: if $a < b$ and $b \leq c$, then $a < c$. Property of ordered groups.

```
lemma (in ring1) ring_strict_ord_trans:
  assumes A1: a<b and A2: b≤c
  shows a<c
proof -
```

```
  from A1 A2 have
    group3(R,A,r)
    ⟨a,b⟩ ∈ r ∧ a≠b  ⟨b,c⟩ ∈ r
    using OrdRing_ZF_1_L4 by auto
    then have ⟨a,c⟩ ∈ r ∧ a≠c by (rule group3.OrderedGroup_ZF_1_L4A)
    then show a<c by simp
qed
```

Another version of transitivity for the strict order: if $a \leq b$ and $b < c$, then $a < c$. Property of ordered groups.

```
lemma (in ring1) ring_strict_ord_transit:
  assumes A1: a≤b and A2: b<c
  shows a<c
proof -
  from A1 A2 have
    group3(R,A,r)
    ⟨a,b⟩ ∈ r  ⟨b,c⟩ ∈ r ∧ b≠c
    using OrdRing_ZF_1_L4 by auto
  then have ⟨a,c⟩ ∈ r ∧ a≠c by (rule group3.group_strict_ord_transit)
  then show a<c by simp
qed
```

The ring order is reflexive.

```
lemma (in ring1) ring_ord_refl: assumes a∈R shows a≤a
  using assms OrdRing_ZF_1_L4(4) group3.OrderedGroup_ZF_1_L3 by simp
```

The next lemma shows what happens when one element of an ordered ring is not greater or equal than another.

```
lemma (in ring1) OrdRing_ZF_1_L4A: assumes A1: a∈R  b∈R
  and A2: ¬(a≤b)
  shows b ≤ a  (-a) ≤ (-b)  a≠b
proof -
  from A1 A2 have I:
    group3(R,A,r)
    r {is total on} R
    a ∈ R  b ∈ R  ⟨a, b⟩ ∉ r
    using OrdRing_ZF_1_L4 by auto
  then have ⟨b,a⟩ ∈ r by (rule group3.OrderedGroup_ZF_1_L8)
  then show b ≤ a by simp
  from I have ⟨GroupInv(R,A)(a),GroupInv(R,A)(b)⟩ ∈ r
    by (rule group3.OrderedGroup_ZF_1_L8)
  then show  (-a) ≤ (-b) by simp
  from I show a≠b by (rule group3.OrderedGroup_ZF_1_L8)
qed
```

A special case of `OrdRing_ZF_1_L4A` when one of the constants is 0. This is useful for many proofs by cases.

```
corollary (in ring1) ord_ring_split2: assumes A1: a∈R
```

```
  shows a≤0 ∨ (0≤a ∧ a≠0)
proof -
  { from A1 have  I: a∈R  0∈R
      using Ring_ZF_1_L2 by auto
    moreover assume A2: ¬(a≤0)
    ultimately have 0≤a by (rule OrdRing_ZF_1_L4A)
    moreover from I A2 have a≠0 by (rule OrdRing_ZF_1_L4A)
    ultimately have 0≤a ∧ a≠0 by simp}
  then show thesis by auto
qed
```

Taking minus on both sides reverses an inequality.

```
lemma (in ring1) OrdRing_ZF_1_L4B: assumes a≤b
  shows (-b) ≤ (-a)
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5
  by simp
```

The next lemma just expands the condition that requires the set of non-negative elements to be closed with respect to multiplication. These are properties of totally ordered groups.

```
lemma (in  ring1) OrdRing_ZF_1_L5:
  assumes 0≤a  0≤b
  shows 0 ≤ a·b
  using pos_mult_closed assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L2
  IsOpClosed_def by simp
```

Double nonnegative is nonnegative.

```
lemma (in  ring1) OrdRing_ZF_1_L5A: assumes A1: 0≤a
  shows 0≤2·a
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5G
  OrdRing_ZF_1_L3 Ring_ZF_1_L3 by simp
```

A sufficient (somewhat redundant) condition for a structure to be an ordered ring. It says that a commutative ring that is a totally ordered group with respect to the additive operation such that set of nonnegative elements is closed under multiplication, is an ordered ring.

```
lemma OrdRing_ZF_1_L6:
  assumes
  IsAring(R,A,M)
  M {is commutative on} R
  Nonnegative(R,A,r) {is closed under} M
  IsAnOrdGroup(R,A,r)
  r {is total on} R
  shows IsAnOrdRing(R,A,M,r)
  using assms IsAnOrdGroup_def Order_ZF_1_L3 IsAnOrdRing_def
  by simp
```

$a \leq b$ iff $a - b \leq 0$. This is a fact from `OrderedGroup.thy`, where it is stated in multiplicative notation.

```
lemma (in ring1) OrdRing_ZF_1_L7:
  assumes a∈R  b∈R
  shows a≤b ⟷ a-b ≤ 0
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9
  by simp
```

Negative times positive is negative.

```
lemma (in ring1) OrdRing_ZF_1_L8:
  assumes A1: a≤0  and A2: 0≤b
  shows a·b ≤ 0
proof -
  from A1 A2 have T1: a∈R  b∈R  a·b ∈ R
    using OrdRing_ZF_1_L3 Ring_ZF_1_L4 by auto
  from A1 A2 have 0≤(-a)·b
    using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5A OrdRing_ZF_1_L5
    by simp
  with T1 show a·b ≤ 0
    using Ring_ZF_1_L7 OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5AA
    by simp
qed
```

We can multiply both sides of an inequality by a nonnegative ring element. This property is sometimes (not here) used to define ordered rings.

```
lemma (in ring1) OrdRing_ZF_1_L9:
  assumes A1: a≤b and A2: 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  from A1 A2 have T1:
    a∈R  b∈R  c∈R  a·c ∈ R  b·c ∈ R
    using OrdRing_ZF_1_L3 Ring_ZF_1_L4 by auto
  with A1 A2 have (a-b)·c ≤ 0
    using OrdRing_ZF_1_L7 OrdRing_ZF_1_L8 by simp
  with T1 show a·c ≤ b·c
    using Ring_ZF_1_L8 OrdRing_ZF_1_L7 by simp
  with mult_commut T1 show c·a ≤ c·b
    using IsCommutative_def by simp
qed
```

A special case of `OrdRing_ZF_1_L9`: we can multiply an inequality by a positive ring element.

```
lemma (in ring1) OrdRing_ZF_1_L9A:
  assumes A1: a≤b and A2: c∈R+
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
```

```
  from A2 have 0 ≤ c using PositiveSet_def
    by simp
  with A1 show a·c ≤ b·c  c·a ≤ c·b
    using OrdRing_ZF_1_L9 by auto
qed
```

A square is nonnegative.

```
lemma (in ring1) OrdRing_ZF_1_L10:
  assumes A1: a∈R shows 0≤(a²)
proof -
  { assume 0≤a
    then have 0≤(a²) using OrdRing_ZF_1_L5 by simp}
  moreover
  { assume ¬(0≤a)
    with A1 have 0≤((-a)²)
      using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L8A
 OrdRing_ZF_1_L5 by simp
    with A1 have 0≤(a²) using Ring_ZF_1_L14 by simp }
  ultimately show thesis by blast
qed
```

1 is nonnegative.

```
corollary (in ring1) ordring_one_is_nonneg: shows 0 ≤ 1
proof -
  have 0 ≤ (1²) using Ring_ZF_1_L2 OrdRing_ZF_1_L10
    by simp
  then show 0 ≤ 1 using Ring_ZF_1_L2 Ring_ZF_1_L3
    by simp
qed
```

In nontrivial rings one is positive.

```
lemma (in ring1) ordring_one_is_pos: assumes 0≠1
  shows 1 ∈ R₊ 0<1
  using assms Ring_ZF_1_L2 ordring_one_is_nonneg PositiveSet_def
  by simp_all
```

Nonnegative is not negative. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_1_L11: assumes 0≤a
  shows ¬(a≤0 ∧ a≠0)
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5AB
  by simp
```

A negative element cannot be a square.

```
lemma (in ring1) OrdRing_ZF_1_L12:
  assumes A1: a≤0  a≠0
  shows ¬(∃b∈R. a = (b²))
proof -
  { assume ∃b∈R. a = (b²)
```

```
    with A1 have False using OrdRing_ZF_1_L10 OrdRing_ZF_1_L11
      by auto
  } then show thesis by auto
qed
```

If $a \leq b$, then $0 \leq b - a$.

```
lemma (in ring1) OrdRing_ZF_1_L13: assumes a≤b
  shows 0 ≤ b-a
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9D
  by simp
```

If $a < b$, then $0 < b - a$.

```
lemma (in ring1) OrdRing_ZF_1_L14: assumes a≤b  a≠b
  shows
  0 ≤ b-a  0 ≠ b-a
  b-a ∈ R+
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9E
  by auto
```

If the difference is nonnegative, then $a \leq b$.

```
lemma (in ring1) OrdRing_ZF_1_L15:
  assumes a∈R b∈R and 0 ≤ b-a
  shows a≤b
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9F
  by simp
```

A nonnegative number is does not decrease when multiplied by a number greater or equal 1.

```
lemma (in ring1) OrdRing_ZF_1_L16:
  assumes A1: 0≤a and A2: 1≤b
  shows a≤a·b
proof -
  from A1 A2 have T: a∈R  b∈R  a·b ∈ R
    using OrdRing_ZF_1_L3 Ring_ZF_1_L4 by auto
  from A1 A2 have 0 ≤ a·(b-1)
    using OrdRing_ZF_1_L13 OrdRing_ZF_1_L5 by simp
  with T show a≤a·b
    using Ring_ZF_1_L8 Ring_ZF_1_L2 Ring_ZF_1_L3 OrdRing_ZF_1_L15
    by simp
qed
```

We can multiply the right hand side of an inequality between nonnegative ring elements by an element greater or equal 1.

```
lemma (in ring1) OrdRing_ZF_1_L17:
  assumes A1: 0≤a and A2: a≤b and A3: 1≤c
  shows a≤b·c
proof -
  from A1 A2 have 0≤b by (rule ring_ord_transitive)
```

```
  with A3 have b≤b·c using OrdRing_ZF_1_L16
    by simp
  with A2 show a≤b·c by (rule ring_ord_transitive)
qed
```

Strict order is preserved by translations.

```
lemma (in ring1) ring_strict_ord_trans_inv:
  assumes a<b and c∈R
  shows
  a+c < b+c
  c+a < c+b
  using assms OrdRing_ZF_1_L4 group3.group_strict_ord_transl_inv
  by auto
```

We can put an element on the other side of a strict inequality, changing its sign.

```
lemma (in ring1) OrdRing_ZF_1_L18:
  assumes a∈R  b∈R and  a-b < c
  shows a < c+b
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L12B
  by simp
```

We can add the sides of two inequalities, the first of them strict, and we get a strict inequality. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_1_L19:
  assumes a<b and c≤d
  shows a+c < b+d
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L12C
  by simp
```

We can add the sides of two inequalities, the second of them strict and we get a strict inequality. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_1_L20:
  assumes a≤b and c<d
  shows a+c < b+d
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L12D
  by simp
```

In a non-trivial ring one is less than two.

```
lemma (in ring1) one_less_two: assumes 0≠1 shows 1<2
  using assms ordring_one_is_pos Ring_ZF_1_L2(2) ring_strict_ord_trans_inv(1)

    Ring_ZF_1_L3(4) by force
```

In a non-trivial ring two is positive.

```
lemma (in ring1) two_positive: assumes 0≠1 shows 2 ∈ R+ 0<2
  using assms ordring_one_is_pos one_less_two ring_strict_ord_transit
element_pos
    by simp_all
```

45.2 Absolute value for ordered rings

Absolute value is defined for ordered groups as a function that is the identity on the nonnegative set and the negative of the element (the inverse in the multiplicative notation) on the rest. In this section we consider properties of absolute value related to multiplication in ordered rings.

Absolute value of a product is the product of absolute values: the case when both elements of the ring are nonnegative.

```
lemma (in ring1) OrdRing_ZF_2_L1:
  assumes 0≤a 0≤b
  shows |a·b| = |a|·|b|
  using assms OrdRing_ZF_1_L5 OrdRing_ZF_1_L4
    group3.OrderedGroup_ZF_1_L2 group3.OrderedGroup_ZF_3_L2
  by simp
```

The absolue value of an element and its negative are the same.

```
lemma (in ring1) OrdRing_ZF_2_L2: assumes a∈R
  shows |-a| = |a|
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_3_L7A by simp
```

The next lemma states that $|a \cdot (-b)| = |(-a) \cdot b| = |(-a) \cdot (-b)| = |a \cdot b|$.

```
lemma (in ring1) OrdRing_ZF_2_L3:
  assumes a∈R  b∈R
  shows
  |(-a)·b| = |a·b|
  |a·(-b)| = |a·b|
  |(-a)·(-b)| = |a·b|
  using assms Ring_ZF_1_L4 Ring_ZF_1_L7 Ring_ZF_1_L7A
   OrdRing_ZF_2_L2 by auto
```

This lemma allows to prove theorems for the case of positive and negative elements of the ring separately.

```
lemma (in ring1) OrdRing_ZF_2_L4: assumes a∈R and ¬(0≤a)
  shows 0 ≤ (-a)  0≠a
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L8A
  by auto
```

Absolute value of a product is the product of absolute values.

```
lemma (in ring1) OrdRing_ZF_2_L5:
  assumes A1: a∈R b∈R
  shows |a·b| = |a|·|b|
proof -
  { assume A2: 0≤a have |a·b| = |a|·|b|
    proof -
      { assume 0≤b
 with A2 have |a·b| = |a|·|b|
   using OrdRing_ZF_2_L1 by simp }
```

```
      moreover
      { assume ¬(0≤b)
 with A1 A2 have |a·(-b)| = |a|·|-b|
   using OrdRing_ZF_2_L4 OrdRing_ZF_2_L1 by simp
 with A1 have |a·b| = |a|·|b|
   using OrdRing_ZF_2_L2 OrdRing_ZF_2_L3 by simp }
      ultimately show thesis by blast
    qed }
  moreover
  { assume ¬(0≤a)
    with A1 have A3: 0 ≤ (-a)
      using OrdRing_ZF_2_L4 by simp
    have |a·b| = |a|·|b|
    proof -
      { assume 0≤b
 with A3 have |(-a)·b| = |-a|·|b|
   using OrdRing_ZF_2_L1 by simp
 with A1 have |a·b| = |a|·|b|
   using OrdRing_ZF_2_L2 OrdRing_ZF_2_L3 by simp }
      moreover
      { assume ¬(0≤b)
 with A1 A3 have |(-a)·(-b)| = |-a|·|-b|
   using OrdRing_ZF_2_L4 OrdRing_ZF_2_L1 by simp
 with A1 have |a·b| = |a|·|b|
   using OrdRing_ZF_2_L2 OrdRing_ZF_2_L3 by simp }
      ultimately show thesis by blast
    qed }
  ultimately show thesis by blast
qed
```

Triangle inequality. Property of linearly ordered abelian groups.

```
lemma (in ring1) ord_ring_triangle_ineq:  assumes a∈R b∈R
  shows |a+b| ≤ |a|+|b|
  using assms OrdRing_ZF_1_L4 group3.OrdGroup_triangle_ineq
  by simp
```

If $a \leq c$ and $b \leq c$, then $a + b \leq 2 \cdot c$.

```
lemma (in ring1) OrdRing_ZF_2_L6:
  assumes a≤c  b≤c shows a+b ≤ 2·c
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5B
    OrdRing_ZF_1_L3 Ring_ZF_1_L3 by simp
```

45.3 Positivity in ordered rings

This section is about properties of the set of positive elements R_+.

The set of positive elements is closed under ring addition. This is a property of ordered groups, we just reference a theorem from OrderedGroup_ZF theory in the proof.

```
lemma (in ring1) OrdRing_ZF_3_L1: shows R+ {is closed under} A
  using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L13
  by simp
```

Every element of a ring can be either in the postitive set, equal to zero or its opposite (the additive inverse) is in the positive set. This is a property of ordered groups, we just reference a theorem from `OrderedGroup_ZF` theory.

```
lemma (in ring1) OrdRing_ZF_3_L2: assumes a∈R
  shows Exactly_1_of_3_holds (a=0, a∈R+, (-a) ∈ R+)
  using assms OrdRing_ZF_1_L4 group3.OrdGroup_decomp
  by simp
```

If a ring element $a \neq 0$, and it is not positive, then $-a$ is positive.

```
lemma (in ring1) OrdRing_ZF_3_L2A: assumes a∈R  a≠0  a ∉ R+
  shows (-a) ∈  R+
  using assms OrdRing_ZF_1_L4 group3.OrdGroup_cases
  by simp
```

R_+ is closed under multiplication iff the ring has no zero divisors.

```
lemma (in ring1) OrdRing_ZF_3_L3:
  shows (R+ {is closed under} M)⟷ HasNoZeroDivs(R,A,M)
proof
  assume A1: HasNoZeroDivs(R,A,M)
  { fix a b assume a∈R+  b∈R+
    then have 0≤a  a≠0  0≤b  b≠0
      using PositiveSet_def by auto
    with A1 have a·b ∈ R+
      using OrdRing_ZF_1_L5 Ring_ZF_1_L2 OrdRing_ZF_1_L3 Ring_ZF_1_L12
 OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L2A
      by simp
  } then show  R+ {is closed under} M using IsOpClosed_def
    by simp
next assume A2: R+ {is closed under} M
  { fix a b assume A3: a∈R  b∈R  and a≠0  b≠0
    with A2 have |a·b| ∈ R+
      using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_3_L12 IsOpClosed_def
        OrdRing_ZF_2_L5 by simp
    with A3 have a·b ≠ 0
      using PositiveSet_def Ring_ZF_1_L4
 OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_3_L2A
      by auto
  } then show HasNoZeroDivs(R,A,M) using HasNoZeroDivs_def
    by auto
qed
```

Another (in addition to `OrdRing_ZF_1_L6` sufficient condition that defines order in an ordered ring starting from the positive set.

```
theorem (in ring0) ring_ord_by_positive_set:
```

```
  assumes
  A1: M {is commutative on} R and
  A2: P⊆R  P {is closed under} A  0 ∉ P and
  A3: ∀a∈R. a≠0 ⟶ (a∈P) Xor ((-a) ∈ P) and
  A4: P {is closed under} M and
  A5: r = OrderFromPosSet(R,A,P)
  shows
  IsAnOrdGroup(R,A,r)
  IsAnOrdRing(R,A,M,r)
  r {is total on} R
  PositiveSet(R,A,r) = P
  Nonnegative(R,A,r) = P ∪ {0}
  HasNoZeroDivs(R,A,M)
proof -
  from A2 A3 A5 show
    I: IsAnOrdGroup(R,A,r)  r {is total on} R and
    II: PositiveSet(R,A,r) = P and
    III: Nonnegative(R,A,r) = P ∪ {0}
    using Ring_ZF_1_L1 group0.Group_ord_by_positive_set
    by auto
  from A2 A4 III have Nonnegative(R,A,r) {is closed under} M
    using Ring_ZF_1_L16 by simp
  with ringAssum A1 I show IsAnOrdRing(R,A,M,r)
    using OrdRing_ZF_1_L6 by simp
  with A4 II show HasNoZeroDivs(R,A,M)
    using OrdRing_ZF_1_L2 ring1.OrdRing_ZF_3_L3
    by auto
qed
```

Nontrivial ordered rings are infinite. More precisely we assume that the neutral element of the additive operation is not equal to the multiplicative neutral element and show that the the set of positive elements of the ring is not a finite subset of the ring and the ring is not a finite subset of itself.

```
theorem (in ring1) ord_ring_infinite: assumes 0≠1
  shows
  R+ ∉ Fin(R)
  R ∉ Fin(R)
  using assms Ring_ZF_1_L17 OrdRing_ZF_1_L4 group3.Linord_group_infinite
  by auto
```

If every element of a nontrivial ordered ring can be dominated by an element from B, then we B is not bounded and not finite.

```
lemma (in ring1) OrdRing_ZF_3_L4:
  assumes 0≠1 and ∀a∈R. ∃b∈B. a≤b
  shows
  ¬IsBoundedAbove(B,r)
  B ∉ Fin(R)
  using assms Ring_ZF_1_L17 OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_2_L2A
  by auto
```

If m is greater or equal the multiplicative unit, then the set $\{m \cdot n : n \in R\}$ is infinite (unless the ring is trivial).

```
lemma (in ring1) OrdRing_ZF_3_L5: assumes A1: 0≠1 and A2: 1≤m
  shows
  {m·x. x∈R+} ∉ Fin(R)
  {m·x. x∈R} ∉ Fin(R)
  {(-m)·x. x∈R} ∉ Fin(R)
proof -
  from A2 have T: m∈R using OrdRing_ZF_1_L3 by simp
  from A2 have 0≤1  1≤m
    using ordring_one_is_nonneg by auto
  then have I: 0≤m by (rule ring_ord_transitive)
  let B = {m·x. x∈R+}
  { fix a assume A3: a∈R
    then have a≤0 ∨ (0≤a ∧ a≠0)
      using ord_ring_split2 by simp
    moreover
    { assume A4: a≤0
      from A1 have m·1 ∈ B using ordring_one_is_pos
 by auto
      with T have m∈B using Ring_ZF_1_L3 by simp
      moreover from A4 I have a≤m by (rule ring_ord_transitive)
      ultimately have ∃b∈B. a≤b by blast }
    moreover
    { assume A4: 0≤a ∧ a≠0
      with A3 have m·a ∈ B using PositiveSet_def
 by auto
      moreover
      from A2 A4 have 1·a ≤ m·a using OrdRing_ZF_1_L9
 by simp
      with A3 have a ≤ m·a using Ring_ZF_1_L3
 by simp
      ultimately have ∃b∈B. a≤b by auto }
    ultimately have ∃b∈B. a≤b by auto
  } then have ∀a∈R. ∃b∈B. a≤b
    by simp
  with A1 show B ∉ Fin(R) using OrdRing_ZF_3_L4
    by simp
  moreover have B ⊆ {m·x. x∈R}
    using PositiveSet_def by auto
  ultimately show {m·x. x∈R} ∉ Fin(R) using Fin_subset
    by auto
  with T show {(-m)·x. x∈R} ∉ Fin(R) using Ring_ZF_1_L18
    by simp
qed
```

If m is less or equal than the negative of multiplicative unit, then the set $\{m \cdot n : n \in R\}$ is infinite (unless the ring is trivial).

```
lemma (in ring1) OrdRing_ZF_3_L6: assumes A1: 0≠1 and A2: m ≤ -1
```

```
  shows {m·x. x∈R} ∉ Fin(R)
proof -
  from A2 have (-(-1)) ≤ -m
    using OrdRing_ZF_1_L4B by simp
  with A1 have {(-m)·x. x∈R} ∉ Fin(R)
    using Ring_ZF_1_L2 Ring_ZF_1_L3 OrdRing_ZF_3_L5
    by simp
  with A2 show {m·x. x∈R} ∉ Fin(R)
    using OrdRing_ZF_1_L3 Ring_ZF_1_L18 by simp
qed
```

All elements greater or equal than an element of R_+ belong to R_+. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_3_L7: assumes A1: a ∈ R₊ and A2: a≤b
  shows b ∈ R₊
proof -
  from A1 A2 have
    group3(R,A,r)
    a ∈ PositiveSet(R,A,r)
    ⟨a,b⟩ ∈ r
    using OrdRing_ZF_1_L4 by auto
  then have b ∈ PositiveSet(R,A,r)
    by (rule group3.OrderedGroup_ZF_1_L19)
  then show b ∈ R₊ by simp
qed
```

A special case of `OrdRing_ZF_3_L7`: a ring element greater or equal than 1 is positive.

```
corollary (in ring1) OrdRing_ZF_3_L8: assumes A1: 0≠1 and A2: 1≤a
  shows a ∈ R₊
proof -
  from A1 A2 have 1 ∈ R₊  1≤a
    using ordring_one_is_pos by auto
  then show a ∈ R₊ by (rule OrdRing_ZF_3_L7)
qed
```

Adding a positive element to a strictly increases a. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_3_L9: assumes A1: a∈R  b∈R₊
  shows a ≤ a+b  a ≠ a+b
  using assms OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L22
  by auto
```

A special case of `OrdRing_ZF_3_L9`: in nontrivial rings adding one to a increases a.

```
corollary (in ring1) OrdRing_ZF_3_L10: assumes A1: 0≠1 and A2: a∈R
  shows a ≤ a+1  a ≠ a+1
  using assms ordring_one_is_pos OrdRing_ZF_3_L9
```

```
  by auto
```

If a is not greater than b, then it is strictly less than $b+1$.

```
lemma (in ring1) OrdRing_ZF_3_L11: assumes A1: 0≠1 and A2: a≤b
  shows a< b+1
proof -
  from A1 A2 have I: b < b+1
    using OrdRing_ZF_1_L3 OrdRing_ZF_3_L10 by auto
  with A2 show a< b+1 by (rule ring_strict_ord_transit)
qed
```

For any ring element a the greater of a and 1 is a positive element that is greater or equal than m. If we add 1 to it we get a positive element that is strictly greater than m. This holds in nontrivial rings.

```
lemma (in ring1) OrdRing_ZF_3_L12: assumes A1: 0≠1 and A2: a∈R
  shows
  a ≤ GreaterOf(r,1,a)
  GreaterOf(r,1,a) ∈ R+
  GreaterOf(r,1,a) + 1 ∈ R+
  a ≤ GreaterOf(r,1,a) + 1  a ≠ GreaterOf(r,1,a) + 1
proof -
  from linord have r {is total on} R using IsLinOrder_def
    by simp
  moreover from A2 have 1 ∈ R  a∈R
    using Ring_ZF_1_L2 by auto
  ultimately have
    1 ≤ GreaterOf(r,1,a) and
    I: a ≤ GreaterOf(r,1,a)
    using Order_ZF_3_L2 by auto
  with A1 show
    a ≤ GreaterOf(r,1,a) and
    GreaterOf(r,1,a) ∈ R+
    using OrdRing_ZF_3_L8 by auto
  with A1 show GreaterOf(r,1,a) + 1 ∈ R+
    using ordring_one_is_pos OrdRing_ZF_3_L1 IsOpClosed_def
    by simp
  from A1 I show
    a ≤ GreaterOf(r,1,a) + 1  a ≠ GreaterOf(r,1,a) + 1
    using OrdRing_ZF_3_L11 by auto
qed
```

We can multiply strict inequality by a positive element.

```
lemma (in ring1) OrdRing_ZF_3_L13:
  assumes A1: HasNoZeroDivs(R,A,M) and
  A2: a<b and A3: c∈R+
  shows
  a·c < b·c
  c·a < c·b
```

```
proof -
  from A2 A3 have T: a∈R  b∈R  c∈R  c≠0
    using OrdRing_ZF_1_L3 PositiveSet_def by auto
  from A2 A3 have a·c ≤ b·c using OrdRing_ZF_1_L9A
    by simp
  moreover from A1 A2 T have a·c ≠ b·c
    using Ring_ZF_1_L12A by auto
  ultimately show a·c < b·c by simp
  moreover from mult_commut T have a·c = c·a and b·c = c·b
    using IsCommutative_def by auto
  ultimately show c·a < c·b by simp
qed
```

A sufficient condition for an element to be in the set of positive ring elements.

```
lemma (in ring1) OrdRing_ZF_3_L14: assumes 0≤a and a≠0
  shows a ∈ R₊
  using assms OrdRing_ZF_1_L3 PositiveSet_def
  by auto
```

If a ring has no zero divisors, the square of a nonzero element is positive.

```
lemma (in ring1) OrdRing_ZF_3_L15:
  assumes HasNoZeroDivs(R,A,M) and a∈R  a≠0
  shows 0 ≤ a²  a² ≠ 0  a² ∈ R₊
  using assms OrdRing_ZF_1_L10 Ring_ZF_1_L12 OrdRing_ZF_3_L14
  by auto
```

In rings with no zero divisors we can (strictly) increase a positive element by multiplying it by an element that is greater than 1.

```
lemma (in ring1) OrdRing_ZF_3_L16:
  assumes HasNoZeroDivs(R,A,M) and a ∈ R₊ and 1≤b  1≠b
  shows a≤a·b  a ≠ a·b
  using assms PositiveSet_def OrdRing_ZF_1_L16 OrdRing_ZF_1_L3
    Ring_ZF_1_L12C by auto
```

If the right hand side of an inequality is positive we can multiply it by a number that is greater than one.

```
lemma (in ring1) OrdRing_ZF_3_L17:
   assumes A1: HasNoZeroDivs(R,A,M) and A2: b∈R₊ and
  A3: a≤b  and A4: 1<c
  shows a<b·c
proof -
  from A1 A2 A4 have b < b·c
    using OrdRing_ZF_3_L16 by auto
  with A3 show a<b·c by (rule ring_strict_ord_transit)
qed
```

We can multiply a right hand side of an inequality between positive numbers by a number that is greater than one.

```
lemma (in ring1) OrdRing_ZF_3_L18:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a ∈ R+ and
  A3: a≤b and A4: 1<c
  shows a<b·c
proof -
  from A2 A3 have b ∈ R+ using OrdRing_ZF_3_L7
    by blast
  with A1 A3 A4 show a<b·c
    using OrdRing_ZF_3_L17 by simp
qed
```

In ordered rings with no zero divisors if at least one of a, b is not zero, then $0 < a^2 + b^2$, in particular $a^2 + b^2 \neq 0$.

```
lemma (in ring1) OrdRing_ZF_3_L19:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a∈R  b∈R and
  A3: a ≠ 0 ∨ b ≠ 0
  shows 0 < a² + b²
proof -
  { assume a ≠ 0
    with A1 A2 have 0 ≤ a²  a² ≠ 0
      using OrdRing_ZF_3_L15 by auto
    then have 0 < a² by auto
    moreover from A2 have 0 ≤ b²
      using OrdRing_ZF_1_L10 by simp
    ultimately have 0 + 0 < a² + b²
      using OrdRing_ZF_1_L19 by simp
    then have 0 < a² + b²
      using Ring_ZF_1_L2 Ring_ZF_1_L3 by simp }
  moreover
  { assume A4: a = 0
    then have a² + b² = 0 + b²
      using  Ring_ZF_1_L2 Ring_ZF_1_L6 by simp
    also from A2 have ... = b²
      using Ring_ZF_1_L4 Ring_ZF_1_L3 by simp
    finally have a² + b² = b² by simp
    moreover
    from A3 A4 have b ≠ 0 by simp
    with A1 A2 have 0 ≤ b² and b² ≠ 0
      using OrdRing_ZF_3_L15 by auto
    hence 0 < b² by auto
    ultimately have 0 < a² + b² by simp }
  ultimately show 0 < a² + b²
    by auto
qed

end
```

46 Groups 4

theory Group_ZF_4 **imports** Group_ZF_1 Group_ZF_2 Finite_ZF Cardinal_ZF

begin

This theory file deals with normal subgroup test and some finite group theory. Then we define group homomorphisms and prove that the set of endomorphisms forms a ring with unity and we also prove the first isomorphism theorem.

46.1 Conjugation of subgroups

First we show some properties of conjugation

The conjugate of a subgroup is a subgroup.

```
theorem (in group0) conj_group_is_group:
  assumes IsAsubgroup(H,P) g∈G
  shows IsAsubgroup({g·(h·g⁻¹). h∈H},P)
proof-
  have sub:H⊆G using assms(1) group0_3_L2 by auto
  from assms(2) have g⁻¹∈G using inverse_in_group by auto
  {
    fix r assume r∈{g·(h·g⁻¹). h∈H}
    then  obtain h where h:h∈H r=g·(h·(g⁻¹)) by auto
    from h(1) have h⁻¹∈H using group0_3_T3A assms(1) by auto
    from h(1) sub have h∈G by auto
    then have h⁻¹∈G using inverse_in_group by auto
    with ‹g⁻¹∈G› have ((h⁻¹)·(g)⁻¹)∈G using group_op_closed by auto
    from h(2) have r⁻¹=(g·(h·(g⁻¹)))⁻¹ by auto moreover
    from ‹h∈G› ‹g⁻¹∈G› have s:h·(g⁻¹)∈G using group_op_closed by blast
    ultimately have r⁻¹=(h·(g⁻¹))⁻¹·(g)⁻¹ using group_inv_of_two assms(2)
by auto
    moreover
    from ‹h∈G› ‹g⁻¹∈G› have (h·(g⁻¹))⁻¹=(g⁻¹)⁻¹·h⁻¹ using group_inv_of_two
by auto
    moreover have (g⁻¹)⁻¹=g using group_inv_of_inv assms(2) by auto
    ultimately have r⁻¹=(g·(h⁻¹))·(g)⁻¹ by auto
    with ‹h⁻¹∈G›‹g⁻¹∈G› have r⁻¹=g·((h⁻¹)·(g)⁻¹) using group_oper_assoc
assms(2) by auto
    moreover from s assms(2) h(2) have r∈G using group_op_closed by
auto
    moreover note ‹h⁻¹∈H› ultimately have r⁻¹∈{g·(h·g⁻¹). h∈H} r∈G by
auto
  }
  then have ∀r∈{g·(h·g⁻¹). h∈H}. r⁻¹∈{g·(h·g⁻¹). h∈H} and {g·(h·g⁻¹).
h∈H}⊆G by auto moreover
  {
    fix s t assume s:s∈{g·(h·g⁻¹). h∈H} and t:t∈{g·(h·g⁻¹). h∈H}
```

```
    then obtain hs ht where hs:hs∈H s=g·(hs·(g⁻¹)) and ht:ht∈H t=g·(ht·(g⁻¹))
by auto
    from hs(1) have hs∈G using sub by auto
    then have g·hs∈G using group_op_closed assms(2) by auto
    then have (g·hs)⁻¹∈G using inverse_in_group by auto
    from ht(1) have ht∈G using sub by auto
    with ‹g⁻¹:G› have ht·(g⁻¹)∈G using group_op_closed by auto
    from hs(2) ht(2) have s·t=(g·(hs·(g⁻¹)))·(g·(ht·(g⁻¹))) by auto
    moreover from ‹hs∈G› have hs·ht = hs·1·ht using group0_2_L2 by auto
    then have hs·ht = hs·(g⁻¹·g)·ht using group0_2_L6 assms(2) by auto
    then have g·(hs·ht) = g·(hs·(g⁻¹·g)·ht) by auto
    with ‹hs∈G› have g·(hs·ht) = g·((hs·g⁻¹·g)·ht) using group_oper_assoc
      assms(2) inverse_in_group by auto
    with ‹hs∈G› ‹ht∈G› have g·(hs·ht) = g·(hs·g⁻¹·(g·ht)) using group_oper_assoc
      assms(2) inverse_in_group group_op_closed by auto
    with ‹hs∈G› ‹ht∈G› have g·(hs·ht) = g·(hs·g⁻¹)·(g·ht) using group_oper_assoc
      assms(2) inverse_in_group group_op_closed by auto
    then have g·(hs·ht)·g⁻¹ = g·(hs·g⁻¹)·(g·ht)·g⁻¹ by auto
    with ‹hs∈G› ‹ht∈G› have g·((hs·ht)·g⁻¹) = g·(hs·g⁻¹)·(g·ht)·g⁻¹ using
group_oper_assoc
      inverse_in_group assms(2) group_op_closed by auto
    with ‹hs∈G› ‹ht∈G› have g·((hs·ht)·g⁻¹) = (g·(hs·g⁻¹))·(g·(ht·g⁻¹)) us-
ing group_oper_assoc
      inverse_in_group assms(2) group_op_closed by auto
    ultimately have s·t=g·((hs·ht)·(g⁻¹)) by auto moreover
    from hs(1) ht(1) have hs·ht∈H using assms(1) group0_3_L6 by auto
    ultimately have s·t∈{g·(h·g⁻¹). h∈H} by auto
  }
  then have {g·(h·g⁻¹). h∈H} {is closed under}P unfolding IsOpClosed_def
by auto moreover
  from assms(1) have 1∈H using group0_3_L5 by auto
  then have g·(1·g⁻¹)∈{g·(h·g⁻¹). h∈H} by auto
  then have {g·(h·g⁻¹). h∈H}≠0 by auto ultimately
  show thesis using group0_3_T3 by auto
qed
```

Every set is equipollent with its conjugates.

```
theorem (in group0) conj_set_is_eqpoll:
  assumes H⊆G g∈G
  shows H≈{g·(h·g⁻¹). h∈H}
proof-
  have fun:{⟨h,g·(h·g⁻¹)⟩. h∈H}:H→{g·(h·g⁻¹). h∈H} unfolding Pi_def function_def
domain_def by auto
  {
    fix h1 h2 assume h1∈Hh2∈H{⟨h,g·(h·g⁻¹)⟩. h∈H}h1={⟨h,g·(h·g⁻¹)⟩. h∈H}h2
    with fun have g·(h1·g⁻¹)=g·(h2·g⁻¹)h1·g⁻¹∈Gh2·g⁻¹∈Gh1∈Gh2∈G using apply_equality
assms(1)
      group_op_closed inverse_in_group assms(2) by auto
    then have h1·g⁻¹=h2·g⁻¹ using group0_2_L19(2) assms(2) by auto
```

```
    with ‹h1∈G› ‹h2∈G› have h1=h2 using group0_2_L19(1) inverse_in_group
assms(2) by auto
  }
  then have ∀h1∈H. ∀h2∈H. {⟨h,g·(h·g⁻¹)⟩. h∈H}h1={⟨h,g·(h·g⁻¹)⟩. h∈H}h2
⟶ h1=h2 by auto
  with fun have {⟨h,g·(h·g⁻¹)⟩. h∈H}∈inj(H,{g·(h·g⁻¹). h∈H}) unfolding
inj_def by auto moreover
  {
    fix ghg assume ghg∈{g·(h·g⁻¹). h∈H}
    then obtain h where h∈H ghg=g·(h·g⁻¹) by auto
    then have ⟨h,ghg⟩∈{⟨h,g·(h·g⁻¹)⟩. h∈H} by auto
    then have {⟨h,g·(h·g⁻¹)⟩. h∈H}h=ghg using apply_equality fun by auto
    with ‹h∈H› have ∃h∈H. {⟨h,g·(h·g⁻¹)⟩. h∈H}h=ghg by auto
  }
  with fun have {⟨h,g·(h·g⁻¹)⟩. h∈H}∈surj(H,{g·(h·g⁻¹). h∈H}) unfolding
surj_def by auto
  ultimately have {⟨h,g·(h·g⁻¹)⟩. h∈H}∈bij(H,{g·(h·g⁻¹). h∈H}) unfolding
bij_def by auto
  then show thesis unfolding eqpoll_def by auto
qed
```

Every normal subgroup contains its conjugate subgroups.

```
theorem (in group0) norm_group_cont_conj:
  assumes IsAnormalSubgroup(G,P,H) g∈G
  shows {g·(h·g⁻¹). h∈H}⊆H
proof-
  {
    fix r assume r∈{g·(h·g⁻¹). h∈H}
    then obtain h where h:r=g·(h·g⁻¹) h∈H by auto moreover
    from h(2) have h∈G using group0_3_L2 assms(1) unfolding IsAnormalSubgroup_def
by auto moreover
    from assms(2) have g⁻¹∈G using inverse_in_group by auto
    ultimately have r=g·h·g⁻¹ h∈H using group_oper_assoc assms(2) by auto
    then have r∈H using assms unfolding IsAnormalSubgroup_def by auto
  }
  then show {g·(h·g⁻¹). h∈H}⊆H by auto
qed
```

If a subgroup contains all its conjugate subgroups, then it is normal.

```
theorem (in group0) cont_conj_is_normal:
  assumes IsAsubgroup(H,P) ∀g∈G. {g·(h·g⁻¹). h∈H}⊆H
  shows IsAnormalSubgroup(G,P,H)
proof-
  {
    fix h g assume h∈H g∈G
    with assms(2) have g·(h·g⁻¹)∈H by auto
    moreover from ‹g∈G›‹h∈H› have h∈G g⁻¹∈G g∈G using group0_3_L2 assms(1)
inverse_in_group by auto
    ultimately have g·h·g⁻¹∈H using group_oper_assoc by auto
```

```
  }
  then show thesis using assms(1) unfolding IsAnormalSubgroup_def by
auto
qed
```

If a group has only one subgroup of a given order, then this subgroup is normal.

```
corollary (in group0) only_one_equipoll_sub:
  assumes IsAsubgroup(H,P) ∀M. IsAsubgroup(M,P)∧ H≈M ⟶ M=H
  shows IsAnormalSubgroup(G,P,H)
proof-
  {
    fix g assume g:g∈G
    with assms(1) have IsAsubgroup({g·(h·g⁻¹). h∈H},P) using conj_group_is_group
by auto
    moreover
    from assms(1) g have H≈{g·(h·g⁻¹). h∈H} using conj_set_is_eqpoll
group0_3_L2 by auto
    ultimately have {g·(h·g⁻¹). h∈H}=H using assms(2) by auto
    then have {g·(h·g⁻¹). h∈H}⊆H by auto
  }
  then show thesis using cont_conj_is_normal assms(1) by auto
qed
```

The trivial subgroup is then a normal subgroup.

```
corollary (in group0) trivial_normal_subgroup:
  shows IsAnormalSubgroup(G,P,{1})
proof-
  have {1}⊆G using group0_2_L2 by auto
  moreover have {1}≠0 by auto moreover
  {
    fix a b assume a∈{1}b∈{1}
    then have a=1b=1 by auto
    then have P⟨a,b⟩=1·1 by auto
    then have P⟨a,b⟩=1 using group0_2_L2 by auto
    then have P⟨a,b⟩∈{1} by auto
  }
  then have {1}{is closed under}P unfolding IsOpClosed_def by auto
  moreover
  {
    fix a assume a∈{1}
    then have a=1 by auto
    then have a⁻¹=1⁻¹ by auto
    then have a⁻¹=1 using group_inv_of_one by auto
    then have a⁻¹∈{1} by auto
  }
  then have ∀a∈{1}. a⁻¹∈{1} by auto ultimately
  have IsAsubgroup({1},P) using group0_3_T3 by auto moreover
  {
```

```
    fix M assume M:IsAsubgroup(M,P) {1}≈M
    then have one:1∈M M≈{1} using eqpoll_sym group0_3_L5 by auto
    then obtain f where f∈bij(M,{1}) unfolding eqpoll_def by auto
    then have inj:f∈inj(M,{1}) unfolding bij_def by auto
    then have fun:f:M→{1} unfolding inj_def by auto
    {
      fix b assume b:b∈M b≠1
      with ‹1∈M› have fb≠f1 using inj unfolding inj_def by auto
      moreover from fun b(1) have fb∈{1} by (rule apply_type)
      moreover from fun one(1) have f1∈{1} by (rule apply_type)
      ultimately have False by auto
    }
    with ‹1∈M› have M={1} by auto
  }
  ultimately show thesis using only_one_equipoll_sub by auto
qed
```

The whole group is normal as a subgroup

```
lemma (in group0) whole_normal_subgroup:
  shows IsAnormalSubgroup(G,P,G)
proof-
  have G⊆G by auto moreover
  have ∀x∈G. x⁻¹∈G using inverse_in_group by auto moreover
  have G≠0 using group0_2_L2 by auto moreover
  have G{is closed under}P using group_op_closed
    unfolding IsOpClosed_def by auto ultimately
  have IsAsubgroup(G,P) using group0_3_T3 by auto
  moreover
  {
    fix n g assume ng:n∈G g∈G
    then have P ⟨P ⟨g, n⟩, GroupInv(G, P) g⟩ ∈ G
      using group_op_closed inverse_in_group by auto
  }
  ultimately show thesis unfolding IsAnormalSubgroup_def by auto
qed
```

46.2 Simple groups

In this subsection we study the groups that build the rest of the groups: the simple groups.

Since the whole group and the trivial subgroup are always normal, it is natural to define simplicity of groups in the following way:

```
definition
  IsSimple ([_,_]{is a simple group} 89)
  where [G,f]{is a simple group} ≡ IsAgroup(G,f) ∧ (∀M. IsAnormalSubgroup(G,f,M)
⟶ M=G∨M={TheNeutralElement(G,f)})
```

From the definition follows that if a group has no subgroups, then it is simple.

```
corollary (in group0) noSubgroup_imp_simple:
  assumes ∀H. IsAsubgroup(H,P)⟶ H=G∨H={1}
  shows [G,P]{is a simple group}
proof-
  have IsAgroup(G,P) using groupAssum by auto moreover
  {
    fix M assume IsAnormalSubgroup(G,P,M)
    then have IsAsubgroup(M,P) unfolding IsAnormalSubgroup_def by auto
    with assms have M=G∨M={1} by auto
  }
  ultimately show thesis unfolding IsSimple_def by auto
qed
```

We add a context for an abelian group

```
locale abelian_group = group0 +
  assumes isAbelian: P {is commutative on} G
```

Since every subgroup is normal in abelian groups, it follows that commutative simple groups do not have subgroups.

```
corollary (in abelian_group) abelian_simple_noSubgroups:
  assumes [G,P]{is a simple group}
  shows ∀H. IsAsubgroup(H,P)⟶ H=G∨H={1}
proof-
  {
    fix H assume A:IsAsubgroup(H,P)H ≠ {1}
    then have IsAnormalSubgroup(G,P,H) using Group_ZF_2_4_L6(1) groupAssum
isAbelian
      by auto
    with assms(1) A have H=G unfolding IsSimple_def by auto
  }
  then show thesis by auto
qed
```

46.3 Finite groups

This subsection deals with finite groups and their structure

The subgroup of a finite group is finite.

```
lemma (in group0) finite_subgroup:
  assumes Finite(G) IsAsubgroup(H,P)
  shows Finite(H)
  using group0_3_L2 subset_Finite assms by force
```

The space of cosets is also finite. In particular, quotient groups.

```
lemma (in group0) finite_cosets:
  assumes Finite(G) IsAsubgroup(H,P)
  defines r ≡ QuotientGroupRel(G,P,H)
```

```
  shows Finite(G//r)
proof-
  have fun:{⟨g,r{g}⟩. g∈G}:G→(G//r) unfolding Pi_def function_def domain_def
by auto
  {
    fix C assume C:C∈G//r
    have equiv:equiv(G,r) using Group_ZF_2_4_L3 assms(2) unfolding r_def
by auto
    then have refl(G,r) unfolding equiv_def by auto
    with C have C≠0 using EquivClass_1_L5 by auto
    then obtain c where c:c∈C by auto
    with C have r{c}=C using EquivClass_1_L2 equiv by auto
    with c C have ⟨c,C⟩∈{⟨g,r{g}⟩. g∈G} using EquivClass_1_L1 equiv by
auto
    then have {⟨g,r{g}⟩. g∈G}c=C c∈G using apply_equality fun by auto
    then have ∃c∈G. {⟨g,r{g}⟩. g∈G}c=C by auto
  }
  with fun have surj:{⟨g,r{g}⟩. g∈G}∈surj(G,G//r) unfolding surj_def by
auto
  from assms(1) obtain n where n∈nat G≈n unfolding Finite_def by auto
  then have G:G≲n Ord(n) using eqpoll_imp_lepoll by auto
  then have G//r≲G using surj_fun_inv_2 surj by auto
  with G(1) have G//r≲n using lepoll_trans by blast
  with ‹n∈nat› show Finite(G//r) using lepoll_nat_imp_Finite by auto
qed
```

All the cosets are equipollent.

```
lemma (in group0) cosets_equipoll:
  assumes IsAsubgroup(H,P) g1∈Gg2∈G
  defines r ≡ QuotientGroupRel(G,P,H)
  shows r{g1} ≈ r{g2}
proof-
  have equiv:equiv(G,r) using Group_ZF_2_4_L3 assms(1) unfolding r_def
by auto
  from assms(3,2) have GG:(g1⁻¹)·g2∈G using inverse_in_group group_op_closed
by auto
  then have bij:RightTranslation(G,P,(g1⁻¹)·g2)∈bij(G,G) using trans_bij(1)
by auto
  have r{g2}∈G//r using assms(3) unfolding quotient_def by auto
  then have sub2:r{g2}⊆G using EquivClass_1_L1 equiv
      assms(3) by blast
  have r{g1}∈G//r using assms(2) unfolding quotient_def by auto
  then have sub:r{g1}⊆G using EquivClass_1_L1 equiv assms(2) by blast
  with bij have restrict(RightTranslation(G,P,(g1⁻¹)·g2),r{g1})∈bij(r{g1},RightTranslation(
    using restrict_bij unfolding bij_def by auto
  then have r{g1}≈RightTranslation(G,P,(g1⁻¹)·g2)(r{g1}) unfolding eqpoll_def
by auto
  with GG sub have A0:r{g1}≈{RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r{g1}}
    using func_imagedef group0_5_L1(1) by force
```

```
  {
    fix t assume t∈{RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r{g1}}
    then obtain q where q:t=RightTranslation(G,P,(g1⁻¹)·g2)q q∈r{g1}
by auto
    then have ⟨g1,q⟩∈r q∈G using image_iff sub by auto
    then have g1·(q⁻¹)∈H q⁻¹∈G using inverse_in_group unfolding r_def
QuotientGroupRel_def by auto
    from GG q sub have t:t=q·((g1⁻¹)·g2) using group0_5_L2(1) by auto
    then have g2·t⁻¹=g2·(q·((g1⁻¹)·g2))⁻¹ by auto
    with ‹q∈G› GG have g2·t⁻¹=g2·(((g1⁻¹)·g2)⁻¹·q⁻¹) using group_inv_of_two
by auto
    then have g2·t⁻¹=g2·(((g2⁻¹)·g1⁻¹⁻¹)·q⁻¹) using group_inv_of_two inverse_in_group
assms(2)
      assms(3) by auto
    then have g2·t⁻¹=g2·(((g2⁻¹)·g1)·q⁻¹) using group_inv_of_inv assms(2)
by auto moreover
    have (g2⁻¹)·g1∈G using assms(2) inverse_in_group assms(3) group_op_closed
by auto
    with assms(3) ‹q⁻¹∈G› have g2·(((g2⁻¹)·g1)·q⁻¹)=g2·((g2⁻¹)·g1)·q⁻¹ us-
ing group_oper_assoc by auto
    moreover have g2·((g2⁻¹)·g1)=g2·(g2⁻¹)·g1 using assms(2) inverse_in_group
assms(3)
      group_oper_assoc by auto
    then have g2·((g2⁻¹)·g1)=g1 using group0_2_L6 assms(3) group0_2_L2
assms(2) by auto ultimately
    have g2·t⁻¹=g1·q⁻¹ by auto
    with ‹g1·(q⁻¹)∈H› have g2·t⁻¹∈H by auto moreover
    from t ‹q∈G› ‹g2∈G› have t∈G using inverse_in_group assms(2) group_op_closed
by auto
    ultimately have ⟨g2,t⟩∈r unfolding QuotientGroupRel_def r_def us-
ing assms(3) by auto
    then have t∈r{g2} using image_iff assms(4) by auto
  }
  then have A1:{RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r{g1}}⊆r{g2} by auto
  {
    fix t assume t∈r{g2}
    then have ⟨g2,t⟩∈r t∈G using sub2 image_iff by auto
    then have H:g2·t⁻¹∈H unfolding QuotientGroupRel_def r_def by auto
    then have G:g2·t⁻¹∈G using group0_3_L2 assms(1) by auto
    then have g1·(g1⁻¹·(g2·t⁻¹))=g1·g1⁻¹·(g2·t⁻¹) using group_oper_assoc
        assms(2) inverse_in_group by auto
    with G have g1·(g1⁻¹·(g2·t⁻¹))=g2·t⁻¹ using group0_2_L6 assms(2) group0_2_L2
by auto
    with H have HH:g1·(g1⁻¹·(g2·t⁻¹))∈H by auto
    from ‹t∈G› have GGG:t·g2⁻¹∈G using inverse_in_group assms(3) group_op_closed
by auto
    from ‹t∈G› have (t·g2⁻¹)⁻¹=g2⁻¹⁻¹·t⁻¹ using group_inv_of_two inverse_in_group
assms(3) by auto
    also have ...=g2·t⁻¹ using group_inv_of_inv assms(3) by auto
```

```
    finally have (t·g2⁻¹)⁻¹=g2·t⁻¹ by auto
    then have g1⁻¹·(t·g2⁻¹)⁻¹=g1⁻¹·(g2·t⁻¹) by auto
    then have ((t·g2⁻¹)·g1)⁻¹=g1⁻¹·(g2·t⁻¹) using group_inv_of_two GGG
assms(2) by auto
    then have HHH:g1·((t·g2⁻¹)·g1)⁻¹∈H using HH by auto
    from ‹t∈G› have (t·g2⁻¹)·g1∈G using assms(2) inverse_in_group assms(3)
group_op_closed by auto
    with HHH have ⟨g1,(t·g2⁻¹)·g1⟩∈r using assms(2) unfolding r_def QuotientGroupRel_def
by auto
    then have rg1:t·g2⁻¹·g1∈r{g1} using image_iff by auto
    from assms(3) have g2⁻¹:G using inverse_in_group by auto
    from ‹t∈G› have t·g2⁻¹·g1·((g1⁻¹)·g2)=t·(g2⁻¹·g1)·((g1⁻¹)·g2) using group_oper_assoc
inverse_in_group assms(3) assms(2)
      by auto
    also from ‹t∈G› have ...=t·((g2⁻¹·g1)·((g1⁻¹)·g2)) using group_oper_assoc
group_op_closed inverse_in_group assms(3) assms(2)
      by auto
    also from GG ‹g2⁻¹:G› have ...=t·(g2⁻¹·(g1·((g1⁻¹)·g2))) using group_oper_assoc
      assms(2) by auto
    also have ...=t·(g2⁻¹·(g1·(g1⁻¹)·g2)) using group_oper_assoc assms(2)
inverse_in_group assms(3) by auto
    also from ‹t∈G› have ...=t using group0_2_L6 assms(3) group0_2_L6
assms(2) group0_2_L2 assms(3) by auto
    finally have t·g2⁻¹·g1·((g1⁻¹)·g2)=t by auto
    with ‹(t·g2⁻¹)·g1∈G› GG have RightTranslation(G,P,(g1⁻¹)·g2)(t·g2⁻¹·g1)=t
using group0_5_L2(1) by auto
    then have t∈{RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r{g1}} using rg1
by force
  }
  then have r{g2}⊆{RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r{g1}} by blast
  with A1 have r{g2}={RightTranslation(G,P,(g1⁻¹)·g2)t. t∈r{g1}} by auto
  with A0 show thesis by auto
qed
```

The order of a subgroup multiplied by the order of the space of cosets is the order of the group. We only prove the theorem for finite groups.

```
theorem (in group0) Lagrange:
  assumes Finite(G) IsAsubgroup(H,P)
  defines r ≡ QuotientGroupRel(G,P,H)
  shows |G|=|H| #* |G//r|
proof-
  have equiv:equiv(G,r) using Group_ZF_2_4_L3 assms(2) unfolding r_def
by auto
  have r{1} ⊆G unfolding r_def QuotientGroupRel_def by auto

  have ∀aa∈G. aa∈H ⟷ ⟨aa,1⟩∈r using Group_ZF_2_4_L5C unfolding r_def
by auto
  then have ∀aa∈G. aa∈H ⟷ ⟨1,aa⟩∈r using equiv unfolding sym_def equiv_def
    by auto
```

then have ∀aa∈G. aa∈H ⟷ aa∈r{1} using image_iff by auto
  with ‹r{1} ⊆G› have H:H=r{1} using group0_3_L2 assms(2) by blast
  {
    fix c assume c∈(G//r)
    then obtain g where g∈G c=r{g} unfolding quotient_def by auto
    then have c≈r{1} using cosets_equipoll assms(2) group0_2_L2 unfolding r_def by auto
    then have |c|=|H| using H cardinal_cong by auto
  }
  then have ∀c∈(G//r). |c|=|H| by auto moreover
  have Finite(G//r) using assms finite_cosets by auto moreover
  have ⋃(G//r)=G using Union_quotient Group_ZF_2_4_L3 assms(2,3) by auto moreover
  from ‹⋃(G//r)=G› have Finite(⋃(G//r)) using assms(1) by auto moreover
  have ∀c1∈(G//r). ∀c2∈(G//r). c1≠c2 ⟶ c1∩c2=0 using quotient_disj
      equiv by blast ultimately
  show thesis using card_partition by auto
qed

46.4 Subgroups generated by sets

In this section we study the minimal subgroup containing a set

Since G is always a group containing the set, we may take the intersection of all subgroups bigger than the set; and hence the result is the subgroup we searched.

definition (in group0)
  SubgroupGenerated (⟨_⟩$_G$ 80)
  where X⊆G ⟹ ⟨X⟩$_G$ ≡ ⋂{H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)}

Every generated subgroup is a subgroup

theorem (in group0) subgroupGen_is_subgroup:
  assumes X⊆G
  shows IsAsubgroup(⟨X⟩$_G$,P)
proof-
  have restrict(P,G×G)=P using group_oper_fun restrict_idem unfolding Pi_def by auto
  then have IsAsubgroup(G,P) unfolding IsAsubgroup_def using groupAssum by auto
  with assms have G∈{H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)} by auto
  then have {H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)}≠0 by auto
  then show thesis using subgroup_inter SubgroupGenerated_def assms by auto
qed

The generated subgroup contains the original set

theorem (in group0) subgroupGen_contains_set:

```
  assumes X⊆G
  shows X ⊆ ⟨X⟩G
proof-
  have restrict(P,G×G)=P using group_oper_fun restrict_idem unfolding
Pi_def by auto
  then have IsAsubgroup(G,P) unfolding IsAsubgroup_def using groupAssum
by auto
  with assms have G∈{H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)} by auto
  then have {H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)}≠0 by auto
  then show thesis using subgroup_inter SubgroupGenerated_def assms by
auto
qed
```

Given a subgroup that contains a set, the generated subgroup from that set is smaller than this subgroup

```
theorem (in group0) subgroupGen_minimal:
  assumes IsAsubgroup(H,P) X⊆H
  shows ⟨X⟩G ⊆ H
proof-
  from assms have sub:X⊆G using group0_3_L2 by auto
  from assms have H∈{H∈Pow(G). X⊆H ∧ IsAsubgroup(H,P)} using group0_3_L2
by auto
  then show thesis using sub subgroup_inter SubgroupGenerated_def assms
by auto
qed

end
```

47 Groups 5

```
theory Group_ZF_5 imports Group_ZF_4 Ring_ZF Semigroup_ZF

begin
```

When the operation P in the group (G, P) is commutative (i.e. the group the abelian) the space End(G,P) of homomorphisms of a group (G, P) into itself has a nice structure.

47.1 First ring of endomorphisms of an abelian group

In this section we show that for an abelian group (G, P) the space End(G,P) (defined in the Group_ZF_2 theory) forms a ring.

The set of endomorphisms is closed under pointwise addition (derived from the group operation). This is so because the group is abelian.

```
theorem (in abelian_group) end_pointwise_addition:
  assumes f∈End(G,P) g∈End(G,P)
```

```
  defines F ≡ P {lifted to function space over} G
  shows F⟨f,g⟩ ∈ End(G,P)
proof-
  from assms(1,2) have fun: f:G→G g∈G→G unfolding End_def by simp_all
  with assms(3) have fun2: F⟨f,g⟩:G→G
    using monoid0.Group_ZF_2_1_L0 group0_2_L1 by simp
  { fix g₁ g₂ assume g₁∈G g₂∈G
    with isAbelian assms fun have
      (F⟨f,g⟩)(g₁·g₂) = (F⟨f,g⟩)(g₁)·(F⟨f,g⟩)(g₂)
      using Group_ZF_2_1_L3 group_op_closed endomor_eq
        apply_type group0_4_L8(3) Group_ZF_2_1_L3 by simp
  } with fun2 show thesis using eq_endomor by simp
qed
```

The value of a product of endomorphisms on a group element is the product of values.

```
lemma (in abelian_group) end_pointwise_add_val:
  assumes f∈End(G,P) g∈End(G,P) x∈G F = P {lifted to function space over}
G
  shows (InEnd(F,G,P)⟨f,g⟩)(x) = (f(x))·(g(x))
  using assms group_oper_fun monoid.group0_1_L3B func_ZF_1_L4
  unfolding End_def by simp
```

The operation of taking the inverse in an abelian group is an endomorphism.

```
lemma (in abelian_group) end_inverse_group:
  shows GroupInv(G,P) ∈ End(G,P)
  using inverse_in_group group_inv_of_two isAbelian IsCommutative_def

    group0_2_T2 groupAssum Homomor_def
  unfolding End_def IsMorphism_def by simp
```

The set of homomorphisms of an abelian group is an abelian subgroup of the group of functions from a set to a group, under pointwise addition.

```
theorem (in abelian_group) end_addition_group:
  assumes F = P {lifted to function space over} G
  shows IsAgroup(End(G,P),InEnd(F,G,P)) and
    InEnd(F,G,P) {is commutative on} End(G,P)
proof-
  have End(G,P)≠0 using end_comp_monoid(1) monoid0.group0_1_L3A
    unfolding monoid0_def by auto
  moreover have End(G,P) ⊆ G→G unfolding End_def by auto
  moreover from isAbelian assms(1) have End(G,P){is closed under} F

    unfolding IsOpClosed_def using end_pointwise_addition by auto
  moreover from groupAssum assms(1) have
    ∀f∈End(G,P). GroupInv(G→G,F)(f) ∈ End(G,P)
    using monoid0.group0_1_L1 end_composition(1) end_inverse_group
      func_ZF_5_L2 group0_2_T2 Group_ZF_2_1_L6
```

```
    unfolding monoid0_def End_def by force
  ultimately show IsAgroup(End(G,P),InEnd(F,G,P))
    using assms(1) group0.group0_3_T3 Group_ZF_2_1_T2
    unfolding IsAsubgroup_def group0_def by blast
  from assms(1) isAbelian show
    InEnd(F,G,P) {is commutative on} End(G,P)
    using Group_ZF_2_1_L7 unfolding End_def IsCommutative_def by auto
qed
```

Endomorphisms form a subgroup of the space of functions that map the group to itself.

```
lemma (in abelian_group) end_addition_subgroup:
  shows IsAsubgroup(End(G,P),P {lifted to function space over} G)
  using end_addition_group unfolding IsAsubgroup_def by simp
```

The neutral element of the group of endomorphisms of a group is the constant function with value equal to the neutral element of the group.

```
lemma (in abelian_group) end_add_neut_elem:
  assumes F = P {lifted to function space over} G
  shows TheNeutralElement(End(G,P),InEnd(F,G,P)) = ConstantFunction(G,1)
  using assms end_addition_subgroup lift_group_subgr_neut by simp
```

For the endomorphisms of a group G the group operation lifted to the function space over G is distributive with respect to the composition operation.

```
lemma (in abelian_group) distributive_comp_pointwise:
  assumes F = P {lifted to function space over} G
  shows
    IsDistributive(End(G,P),InEnd(F,G,P),InEnd(Composition(G),G,P))
proof -
  let C_G = Composition(G)
  let C_E = InEnd(C_G,G,P)
  let F_E = InEnd(F,G,P)
  { fix b c d assume AS: b∈End(G,P) c∈End(G,P) d∈End(G,P)
    with assms(1) have ig1: C_G ⟨b, F  ⟨c, d⟩⟩ = b O (F⟨c,d⟩)
      using monoid.Group_ZF_2_1_L0 func_ZF_5_L2 unfolding End_def
      by auto
    with AS have ig2: F⟨C_G⟨b,c⟩,C_G ⟨b,d⟩⟩ = F⟨b O c,b O d⟩
      unfolding End_def using func_ZF_5_L2 by auto
    from assms(1) AS have comp1fun: (b O (F⟨c,d⟩)):G→G
      using monoid.Group_ZF_2_1_L0 comp_fun unfolding End_def by force
    from assms(1) AS have comp2fun: (F ⟨b O c,b O d⟩) : G→G
      using monoid.Group_ZF_2_1_L0 comp_fun unfolding End_def by force
    { fix g assume g∈G
      with assms(1) AS(2,3) have (b O (F⟨c,d⟩))(g) = b((F⟨c,d⟩)(g))
        using comp_fun_apply monoid.Group_ZF_2_1_L0 unfolding End_def

        by force
      with groupAssum assms(1) AS ‹g∈G› have
```

```
    (b O (F⟨c,d⟩))(g) = (F⟨b O c,b O d⟩)(g)
    using Group_ZF_2_1_L3 apply_type homomor_eq comp_fun
    unfolding End_def by auto
} hence ∀g∈G. (b O (F⟨c,d⟩))(g) = (F⟨b O c,b O d⟩)(g) by simp
with comp1fun comp2fun ig1 ig2 have
  C_G⟨b,F⟨c, d⟩⟩ = F⟨C_G⟨b , c⟩,C_G⟨b,d⟩⟩
  using func_eq by simp
moreover from AS(2,3) have F⟨c, d⟩ = F_E⟨c, d⟩
  using restrict by simp
moreover from AS have C_G⟨b,c⟩ = C_E⟨b,c⟩ and C_G⟨b,d⟩ = C_E⟨b,d⟩
  using restrict by auto
moreover from assms AS have C_G⟨b,F ⟨c,d⟩⟩ = C_E⟨b, F⟨c, d⟩⟩
  using end_pointwise_addition by simp
moreover from AS have F⟨C_G⟨b,c⟩,C_G⟨b,d⟩⟩ = F_E⟨C_G ⟨b,c⟩,C_G ⟨b,d⟩⟩
  using end_composition by simp
ultimately have eq1: C_E⟨b, F_E⟨c,d⟩⟩ = F_E ⟨C_E⟨b,c⟩,C_E⟨b,d⟩⟩
  by simp
from assms(1) AS have
  compfun: (F⟨c,d⟩) O b : G→G F⟨c O b,d O b⟩ : G→G
  using monoid.Group_ZF_2_1_L0 comp_fun unfolding End_def by auto
{ fix g assume g∈G
  with AS(1) have bg: b(g) ∈ G unfolding End_def using apply_type

    by auto
  from ‹g∈G› AS(1) have ((F⟨c,d⟩) O b)g = (F⟨c,d⟩)(b(g))
    using comp_fun_apply  unfolding End_def by force
  also from assms(1) AS(2,3) bg have ... = (c(b(g)))·(d(b(g)))
    using Group_ZF_2_1_L3 unfolding End_def by auto
  also from ‹g∈G› AS have ... = ((c O b)(g))·((d O b)(g))
    using comp_fun_apply unfolding End_def by auto
  also from assms(1) ‹g∈G› AS have ... = (F⟨c O b,d O b⟩)g
    using comp_fun Group_ZF_2_1_L3 unfolding End_def by auto
  finally have ((F⟨c,d⟩) O b)(g) = (F⟨c O b,d O b⟩)(g) by simp
} hence ∀g∈G. ((F⟨c,d⟩) O b)(g) = (F⟨c O b,d O b⟩)(g) by simp
with compfun have (F⟨c,d⟩) O b = F⟨c O b,d O b⟩
  using func_eq by blast
with assms(1) AS have C_G⟨F⟨c,d⟩,b⟩ = F⟨C_G⟨c,b⟩,C_G⟨d , b⟩⟩
  using monoid.Group_ZF_2_1_L0 func_ZF_5_L2 unfolding End_def
  by simp
moreover from AS(2,3) have F⟨c, d⟩ = F_E⟨c, d⟩
  using restrict by simp
moreover from AS have C_G⟨c,b⟩ = C_E⟨c , b⟩ C_G⟨d,b⟩ = C_E⟨d,b⟩
  using restrict by auto
moreover from assms AS have C_G⟨F⟨c,d⟩,b⟩ = C_E⟨F⟨c,d⟩,b⟩
  using end_pointwise_addition by auto
moreover from AS have F⟨C_G⟨c,b⟩,C_G⟨d,b⟩⟩ = F_E⟨C_G⟨c,b⟩,C_G⟨d,b⟩⟩
  using end_composition by auto
ultimately have C_E⟨F_E⟨c,d⟩,b⟩ = F_E⟨C_E⟨c,b⟩,C_E⟨d,b⟩⟩
  by auto
```

```
    with eq1 have (C_E⟨b, F_E⟨c, d⟩⟩ = F_E⟨C_E⟨b,c⟩,C_E⟨b,d⟩⟩) ∧
      (C_E⟨F_E⟨c,d⟩,b⟩ = F_E⟨C_E⟨c,b⟩,C_E⟨d,b⟩⟩)
      by auto
  } then show thesis unfolding IsDistributive_def by auto
qed
```

The endomorphisms of an abelian group is in fact a ring with the previous operations.

```
theorem (in abelian_group) end_is_ring:
  assumes F = P {lifted to function space over} G
  shows
    IsAring(End(G,P),InEnd(F,G,P),InEnd(Composition(G),G,P))
  using assms end_addition_group end_comp_monoid(1) distributive_comp_pointwise
  unfolding IsAring_def by auto
```

The theorems proven in the `ring0` context are valid in the `abelian_group` context as applied to the endomorphisms of G.

```
sublocale abelian_group < endo_ring: ring0
  End(G,P)
  InEnd(P {lifted to function space over} G,G,P)
  InEnd(Composition(G),G,P)
  λx b. InEnd(P {lifted to function space over} G,G,P)⟨x,b⟩
  λx. GroupInv(End(G, P), InEnd(P {lifted to function space over} G,G,P))(x)

  λx b. InEnd(P {lifted to function space over} G,G,P)⟨x, GroupInv(End(G,
P), InEnd(P {lifted to function space over} G,G,P))(b)⟩
  λx b. InEnd(Composition(G),G,P)⟨x, b⟩
  TheNeutralElement(End(G, P),InEnd(P {lifted to function space over}
G,G,P))
  TheNeutralElement(End(G, P),InEnd(Composition(G),G,P))
  InEnd(P {lifted to function space over} G,G,P)
     ⟨TheNeutralElement (End(G, P), InEnd(Composition(G),G,P)),
      TheNeutralElement (End(G, P), InEnd(Composition(G),G,P))⟩
  λx. InEnd(Composition(G),G,P)⟨x, x⟩
  λs. Fold(InEnd(P {lifted to function space over} G,G,P), TheNeutralElement(End(G,
P), InEnd(P {lifted to function space over} G,G,P)),s)
  λn x. Fold(InEnd(P {lifted to function space over} G,G,P), TheNeutralElement(End(G,
P), InEnd(P {lifted to function space over} G,G,P)),{⟨k,x⟩. k∈n})
  using end_is_ring unfolding ring0_def by blast
```

47.2 First isomorphism theorem

Now we will prove that any homomorphism $f : G \to H$ defines a bijective homomorphism between G/H and $f(G)$.

A group homomorphism sends the neutral element to the neutral element.

```
lemma image_neutral:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F)
```

```
  shows f(TheNeutralElement(G,P)) = TheNeutralElement(H,F)
proof -
  let eG = TheNeutralElement(G,P)
  let eH = TheNeutralElement(H,F)
  from assms(3) have ff: f:G→H
    unfolding Homomor_def by simp
  have g: eG = P⟨eG,eG⟩ eG ∈ G
    using assms(1) group0.group0_2_L2 unfolding group0_def by simp_all
  with assms have f(eG) = F⟨f(eG),f(eG)⟩
    unfolding Homomor_def IsMorphism_def by force
  moreover
  from ff g(2) have h: f(eG) ∈ H using apply_type
    by simp
  with assms(2) have f(eG) = F⟨f(eG),eH⟩
    using group0.group0_2_L2 unfolding group0_def by simp
  ultimately have F⟨f(eG),eH⟩ = F⟨f(eG),f(eG)⟩
    by simp
  with assms(2) h have
    LeftTranslation(H,F,f(eG))(eH) = LeftTranslation(H,F,f(eG))(f(eG))
    using group0.group0_5_L2(2) group0.group0_2_L2 unfolding group0_def

      by simp
  moreover from assms(2) h have LeftTranslation(H,F,f(eG))∈inj(H,H)
      using group0.trans_bij(2) unfolding group0_def bij_def
      by simp
  ultimately show thesis using h assms(2) group0.group0_2_L2
    unfolding inj_def group0_def by force
qed
```

If $f : G \to H$ is a homomorphism, then it commutes with the inverse

```
lemma image_inv:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F) g∈G
  shows f(GroupInv(G,P)(g)) = GroupInv(H,F)(f(g))
proof -
  from assms(3) have ff: f:G→H
    unfolding Homomor_def by simp
  with assms(4) have im: f(g)∈H using apply_type by simp
  from assms(1,4) have inv: GroupInv(G,P)(g)∈G
    using group0.inverse_in_group unfolding group0_def by simp
  with ff have inv2: f(GroupInv(G,P)g)∈H using apply_type by simp
  from assms(1,4) have
    f(TheNeutralElement(G,P)) = f(P⟨g,GroupInv(G,P)(g)⟩)
    using group0.group0_2_L6 unfolding group0_def by simp
  also from assms inv have ... = F⟨f(g),f(GroupInv(G,P)(g))⟩
    unfolding Homomor_def IsMorphism_def by simp
  finally have f(TheNeutralElement(G,P)) = F⟨f(g),f(GroupInv(G,P)(g))⟩
    by simp
  with assms im inv2 show thesis
    using group0.group0_2_L9 image_neutral unfolding group0_def by simp
```

qed

The preimage of a subgroup is a subgroup

```
theorem preimage_sub:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F)
          IsAsubgroup(K,F)
  shows IsAsubgroup(f-(K),P)
proof -
  from assms(3) have ff: f:G→H
    unfolding Homomor_def by simp
  from assms(2) have Hgr: group0(H,F) unfolding group0_def by simp
  from assms(1) have Ggr: group0(G,P) unfolding group0_def by simp
  moreover
  from assms ff Ggr Hgr have TheNeutralElement(G,P) ∈ f-(K)
    using image_neutral group0.group0_3_L5 func1_1_L15 group0.group0_2_L2

    by simp
  hence f-(K)≠0 by blast
  moreover from ff have f-(K) ⊆ G using func1_1_L3 by simp
  moreover from assms ff Ggr Hgr have f-(K) {is closed under} P
    using func1_1_L15 group0.group0_3_L6 group0.group_op_closed func1_1_L15
    unfolding IsOpClosed_def Homomor_def IsMorphism_def by simp
  moreover from assms ff Ggr Hgr have
    ∀x∈f-(K). GroupInv(G, P)(x) ∈ f-(K)
    using group0.group0_3_T3A image_inv func1_1_L15
        group0.inverse_in_group by simp
  ultimately show thesis by (rule group0.group0_3_T3)
qed
```

The preimage of a normal subgroup is normal

```
theorem preimage_normal_subgroup:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F)
          IsAnormalSubgroup(H,F,K)
        shows IsAnormalSubgroup(G,P,f-(K))
proof -
  from assms(3) have ff: f:G→H
    unfolding Homomor_def by simp
  from assms(2) have Hgr: group0(H,F) unfolding group0_def by simp
  with assms(4) have K⊆H using group0.group0_3_L2
    unfolding IsAnormalSubgroup_def by simp
  from assms(1) have Ggr: group0(G,P) unfolding group0_def by simp
  moreover from assms have IsAsubgroup(f-(K),P)
    using preimage_sub unfolding IsAnormalSubgroup_def by simp
  moreover
  { fix g assume gG: g∈G
    { fix h assume h ∈ {P⟨g,P⟨h, GroupInv(G, P)(g)⟩⟩. h ∈ f-(K)}
      then obtain k where
        k: h = P⟨g,P⟨k,GroupInv(G, P)(g)⟩⟩ k ∈ f-(K)
        by auto
```

```
      from k(1) have f(h) = f(P⟨g,P⟨k, GroupInv(G, P)(g)⟩⟩) by simp
      moreover from ff k(2) have k∈G using vimage_iff
        unfolding Pi_def by blast
      ultimately have f: f(h) = F⟨f(g),F⟨f(k),GroupInv(H,F)(f(g))⟩⟩
        using assms(1-4) Ggr gG group0.group_op_closed
          group0.inverse_in_group image_inv homomor_eq by simp
      from assms(1) ff Ggr ‹g∈G› k have h∈G using group0.group_op_closed
        group0.inverse_in_group func1_1_L15 by simp
      from assms(4) ff k(2) ‹g∈G› have f(k)∈K f(g)∈H and
        F⟨F⟨f(g),f(k)⟩,GroupInv(H,F)(f(g))⟩ ∈ K
        using func1_1_L15 apply_type unfolding IsAnormalSubgroup_def
        by auto
      moreover from ‹f(k)∈K› ‹K⊆H› Hgr f ‹f(g)∈H› have
        f(h) = F⟨F⟨f(g),f(k)⟩,GroupInv(H,F)(f(g))⟩
        using group0.group_oper_assoc group0.inverse_in_group by auto
      ultimately have f(h) ∈ K by simp
      with ff ‹h∈G› have h ∈ f-(K) using func1_1_L15 by simp
    } hence {P⟨g,P⟨h,GroupInv(G,P)(g)⟩⟩. h∈f-(K)} ⊆ f-(K)
      by blast
  } hence ∀g∈G. {P⟨g, P⟨h, GroupInv(G, P)(g)⟩⟩. h∈f-(K)} ⊆ f-(K)
    by simp
  ultimately show thesis using group0.cont_conj_is_normal by simp
qed
```

The kernel of an homomorphism is a normal subgroup.

```
corollary kernel_normal_sub:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F)
  shows IsAnormalSubgroup(G,P,f-{TheNeutralElement(H,F)})
  using assms preimage_normal_subgroup group0.trivial_normal_subgroup

  unfolding group0_def by auto
```

The image of a subgroup is a subgroup

```
theorem image_subgroup:
  assumes IsAgroup(G,P) IsAgroup(H,F)
    Homomor(f,G,P,H,F) f:G→H IsAsubgroup(K,P)
  shows IsAsubgroup(fK,F)
proof -
  from assms(1,5) have sub: K⊆G using group0.group0_3_L2
    unfolding group0_def by simp
  from assms(2) have group0(H,F) unfolding group0_def by simp
  moreover from assms(4) have f(K) ⊆ H
    using func_imagedef sub apply_type by auto
  moreover
  from assms(1,4,5) sub have f(TheNeutralElement(G,P)) ∈ f(K)
    using group0.group0_3_L5 func_imagedef unfolding group0_def
    by auto
  hence f(K) ≠ 0 by blast
  moreover
```

```
  { fix x assume x∈f(K)
    with assms(4) sub obtain q where q: q∈K x=f(q)
      using func_imagedef by auto
    with assms(1-4) sub have GroupInv(H,F)(x) = f(GroupInv(G,P)q)
      using image_inv by auto
    with assms(1,4,5) q(1) sub have GroupInv(H,F)(x) ∈ f(K)
      using group0.group0_3_T3A func_imagedef unfolding group0_def
      by auto
  } hence ∀x∈f(K). GroupInv(H, F)(x) ∈ f(K) by auto
  moreover
  { fix x y assume x∈f(K) y∈f(K)
    with assms(4) sub obtain q_x q_y where
      q: q_x∈K x=f(q_x) q_y∈K y=f(q_y)
      using func_imagedef by auto
    with assms(1-3) sub have F⟨x,y⟩ = f(P⟨q_x,q_y⟩)
      using homomor_eq by force
    moreover from assms(1,5) q(1,3) have P⟨q_x,q_y⟩ ∈ K
      using group0.group0_3_L6 unfolding group0_def by simp
    ultimately have F⟨x,y⟩  ∈ f(K)
      using assms(4) sub func_imagedef by auto
  } then have  f(K) {is closed under} F unfolding IsOpClosed_def
    by simp
  ultimately show thesis using group0.group0_3_T3 by simp
qed
```

The image of a group under a homomorphism is a subgroup of the target group.

```
corollary image_group:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F)
  shows IsAsubgroup(f(G),F)
proof -
  from assms(1) have restrict(P,G×G) = P
    using group0.group_oper_fun restrict_domain unfolding group0_def
    by blast
  with assms show thesis using image_subgroup
    unfolding Homomor_def IsAsubgroup_def by simp
qed
```

Now we are able to prove the first isomorphism theorem. This theorem states that any group homomorphism $f : G \to H$ gives an isomorphism between a quotient group of G and a subgroup of H.

```
theorem isomorphism_first_theorem:
  assumes IsAgroup(G,P) IsAgroup(H,F) Homomor(f,G,P,H,F)
  defines r ≡ QuotientGroupRel(G,P,f-{TheNeutralElement(H,F)}) and
  𝒫 ≡ QuotientGroupOp(G,P,f-{TheNeutralElement(H,F)})
  shows ∃𝔣. Homomor(𝔣,G//r,𝒫,f(G),restrict(F,(f(G))×(f(G)))) ∧ 𝔣∈bij(G//r,f(G))
proof-
  let 𝔣 = {⟨r{g},f(g)⟩. g∈G}
  from assms(3) have ff: f:G→H
```

```
    unfolding Homomor_def by simp
  from assms(1-5) have equiv(G,r)
    using group0.Group_ZF_2_4_L3 kernel_normal_sub
    unfolding group0_def IsAnormalSubgroup_def by simp
  from assms(4) ff have 𝔣 ∈ Pow((G//r)×f(G))
    unfolding quotient_def using func_imagedef by auto
  moreover have (G//r) ⊆ domain(𝔣) unfolding domain_def quotient_def
by auto
  moreover
  { fix x y t assume A: ⟨x,y⟩ ∈ 𝔣 ⟨x,t⟩ ∈ 𝔣
    then obtain g_y g_r where ⟨x, y⟩=⟨r{g_y},f(g_y)⟩ ⟨x, t⟩=⟨r{g_r},f(g_r)⟩
      and g_r∈G g_y∈G by auto
    hence B: r{g_y}=r{g_r} y=f(g_y) t=f(g_r) by auto
    from ff ‹g_y∈G› ‹g_r∈G› B(2,3) have y∈H t∈H
      using apply_type by simp_all
    with ‹equiv(G,r)› ‹g_r∈G› ‹r{g_y}=r{g_r}› have ⟨g_y,g_r⟩∈r
      using same_image_equiv by simp
    with assms(4) ff have
      f(P⟨g_y,GroupInv(G,P)(g_r)⟩) = TheNeutralElement(H,F)
      unfolding QuotientGroupRel_def using func1_1_L15 by simp
    with assms(1-4) B(2,3) ‹g_y∈G› ‹g_r∈G› ‹y∈H› ‹t∈H› have y=t
      using image_inv group0.inverse_in_group group0.group0_2_L11A
      unfolding group0_def Homomor_def IsMorphism_def by auto
  } hence ∀x y. ⟨x,y⟩ ∈ 𝔣 ⟶ (∀z. ⟨x,z⟩∈𝔣 ⟶ y=z) by auto
  ultimately have ff_fun: 𝔣:G//r→f(G) unfolding Pi_def function_def
    by auto
  { fix a₁ a₂ assume AS: a₁∈G//r a₂∈G//r
    then obtain g₁ g₂  where g₁∈G g₂∈G and a: a₁=r{g₁} a₂=r{g₂}
      unfolding quotient_def by auto
    with assms ‹equiv(G,r)› have ⟨𝒫⟨a₁,a₂⟩,f(P⟨g₁,g₂⟩)⟩ ∈ 𝔣
      using Group_ZF_2_4_L5A kernel_normal_sub group0.Group_ZF_2_2_L2
group0.group_op_closed
      unfolding QuotientGroupOp_def group0_def by auto
    with ff_fun have eq: 𝔣(𝒫⟨a₁,a₂⟩) = f(P⟨g₁,g₂⟩) using apply_equality

      by simp
    from ‹g₁∈G› ‹g₂∈G› a have ⟨a₁,f(g₁)⟩ ∈ 𝔣 and ⟨a₂,f(g₂)⟩ ∈ 𝔣 by auto
    with assms(1,2,3) ff_fun ‹g₁∈G› ‹g₂∈G› eq have F⟨𝔣(a₁),𝔣(a₂)⟩ = 𝔣(𝒫⟨a₁,a₂⟩)
      using apply_equality unfolding Homomor_def IsMorphism_def by simp
    moreover from AS ff_fun have 𝔣(a₁) ∈ f(G) 𝔣(a₂) ∈ f(G)
      using apply_type by auto
    ultimately have restrict(F,f(G)×f(G))⟨𝔣(a₁),𝔣(a₂)⟩ = 𝔣(𝒫⟨a₁,a₂⟩)
      by simp
  } hence
    r: ∀a₁∈G//r. ∀a₂∈G//r. restrict(F,f(G)×f(G))⟨𝔣(a₁),𝔣(a₂)⟩ = 𝔣(𝒫⟨a₁,a₂⟩)

    by simp
  with ff_fun have HOM: Homomor(𝔣,G//r,𝒫,f(G),restrict(F,(f(G))×(f(G))))
    unfolding Homomor_def IsMorphism_def by simp
```

```
  from assms have G: IsAgroup(G//r,𝒫) and
      H: IsAgroup(f(G), restrict(F,f(G)×f(G)))
        using Group_ZF_2_4_T1 kernel_normal_sub image_group
        unfolding IsAsubgroup_def by simp_all
  { fix b₁ b₂ assume AS: 𝔣(b₁) = 𝔣(b₂) b₁∈G//r b₂∈G//r
    from G AS(3) have invb2: GroupInv(G//r,𝒫)(b₂)∈G//r
      using group0.inverse_in_group unfolding group0_def by simp
    with G AS(2) have I: 𝒫⟨b₁,GroupInv(G//r,𝒫)(b₂)⟩∈G//r
      using group0.group_op_closed unfolding group0_def by auto
    then obtain g where g∈G and gg: 𝒫⟨b₁,GroupInv(G//r,𝒫)(b₂)⟩=r{g}

      unfolding quotient_def by auto
    from ‹g∈G› have ⟨r{g},f(g)⟩ ∈ 𝔣 by blast
    with ff_fun gg have E: 𝔣(𝒫⟨b₁,GroupInv(G//r,𝒫)(b₂)⟩) = f(g)
      using apply_equality by simp
    from ff_fun invb2 have pp: 𝔣(GroupInv(G//r,𝒫)(b₂))∈f(G)
      using apply_type by simp
    from ff_fun AS(2,3) have fff: 𝔣(b₁) ∈ f(G) 𝔣(b₂) ∈ f(G)
      using apply_type by simp_all
    from fff(1) pp have
      EE: F⟨𝔣(b₁),𝔣(GroupInv(G//r,𝒫)(b₂))⟩=
          restrict(F,f(G)×f(G))⟨𝔣(b₁),𝔣(GroupInv(G//r,𝒫)(b₂))⟩
      by auto
    from ff have f(G) ⊆ H using func1_1_L6(2) by simp
    with fff have 𝔣(b₁)∈H 𝔣(b₂)∈H by auto
    with assms(1-4) G H HOM ff_fun AS(1,3) fff(2) EE have
      TheNeutralElement(H,F) =
        restrict(F,f(G)×f(G))⟨𝔣(b₁),𝔣(GroupInv(G//r,𝒫)(b₂))⟩
      using group0.group0_2_L6(1) restrict image_inv group0.group0_3_T1
image_group
      unfolding group0_def by simp
    also from G H HOM AS(2,3) E have ... = f(g)
      using group0.inverse_in_group unfolding group0_def IsMorphism_def
Homomor_def
      by simp
    finally have TheNeutralElement(H,F) = f(g) by simp
    with ff ‹g∈G› have g∈f-{TheNeutralElement(H,F)} using func1_1_L15

      by simp
    with assms ‹g∈G› gg have
      𝒫⟨b₁,GroupInv(G//r,𝒫)(b₂)⟩ = TheNeutralElement(G//r,𝒫)
      using group0.Group_ZF_2_4_L5E kernel_normal_sub unfolding group0_def

      by simp
    with AS(2,3) G have b₁=b₂ using group0.group0_2_L11A unfolding group0_def

      by auto
  } with ff_fun have 𝔣 ∈ inj(G//r,f(G)) unfolding inj_def by blast
  moreover
```

```
  { fix m assume m ∈ f(G)
    with ff obtain g where g∈G m=f(g) using func_imagedef by auto
    hence ⟨r{g},m⟩ ∈ 𝔣 by blast
    with ff_fun have 𝔣(r{g})=m using apply_equality by auto
    with ‹g∈G› have ∃A∈G//r. 𝔣(A) = m unfolding quotient_def by auto
  }
  ultimately have 𝔣 ∈ bij(G//r,fG) unfolding bij_def surj_def
    using ff_fun by blast
  with HOM show thesis by blast
qed
```

The inverse of a bijective homomorphism is an homomorphism. Meaning that in the previous result, the homomorphism we found is an isomorphism.

```
theorem bij_homomor:
  assumes f∈bij(G,H) IsAgroup(G,P) Homomor(f,G,P,H,F)
  shows Homomor(converse(f),H,F,G,P)
proof -
  { fix h1 h2 assume h1∈H h2∈H
    with assms(1) obtain g1 g2 where
      g1: g1∈G f(g1)=h1 and g2: g2∈G f(g2)=h2
      unfolding bij_def surj_def by blast
    with assms(2,3) have
      converse(f)(f(P⟨g1,g2⟩)) = converse(f)(F⟨h1,h2⟩)
      using homomor_eq by simp
    with assms(1,2) g1 g2 have
      P⟨converse(f)(h1),converse(f)(h2)⟩ = converse(f)(F⟨h1,h2⟩)
      using left_inverse group0.group_op_closed unfolding group0_def bij_def
      by auto
  } with assms(1) show thesis using bij_converse_bij bij_is_fun
    unfolding Homomor_def IsMorphism_def by simp
qed
```

A very important homomorphism is given by taking every element to its class in a group quotient. Recall that $\lambda x \in X.p(x)$ is an alternative notation for function defined as a set of pairs, see lemma `lambda_fun_alt` in theory `func1.thy`.

```
lemma (in group0) quotient_map:
  assumes IsAnormalSubgroup(G,P,H)
  defines r ≡ QuotientGroupRel(G,P,H) and q ≡ λx∈G. QuotientGroupRel(G,P,H){x}
  shows Homomor(q,G,P,G//r,QuotientGroupOp(G,P,H))
  using groupAssum assms group_op_closed lam_funtype lamE EquivClass_1_L10

    Group_ZF_2_4_L3 Group_ZF_2_4_L5A  Group_ZF_2_4_T1
  unfolding IsAnormalSubgroup_def QuotientGroupOp_def Homomor_def IsMorphism_def
  by simp
```

In the context of `group0`, we may use all results of `semigr0`.

```
sublocale group0 < semigroup:semigr0 G P groper λx. Fold1(P,x) Append
Concat
```

unfolding semigr0_def **using** groupAssum IsAgroup_def IsAmonoid_def **by** auto

end

48 Rings - Ideals

theory Ring_ZF_2 **imports** Ring_ZF Group_ZF_2 Finite_ZF Finite1 Cardinal_ZF Semigroup_ZF

begin

This section defines the concept of a ring ideal, and defines some basic concepts and types, finishing with the theorem that shows that the quotient of the additive group by the ideal is actually a full ring.

48.1 Ideals

In ring theory ideals are special subsets of a ring that play a similar role as normal subgroups in the group theory.

An ideal is a subgroup of the additive group of the ring, which is closed by left and right multiplication by any ring element.

```
definition (in ring0) Ideal (_◁R) where
  I ◁R ≡ (∀x∈I. ∀y∈R. y·x∈I ∧ x·y∈I) ∧ IsAsubgroup(I,A)
```

To write less during proofs, we will write $\mathcal{I}$ to denote the set of ideals of the ring R.

```
abbreviation (in ring0) ideals (ℐ) where ℐ ≡ {J∈Pow(R). J◁R}
```

The first examples of ideals are the whole ring and the zero ring:

```
lemma (in ring0) ring_self_ideal:
  shows R ◁R
  using add_group.group_self_subgroup Ring_ZF_1_L4(3)
  unfolding Ideal_def by simp
```

The singleton containing zero is and ideal.

```
lemma (in ring0) zero_ideal:
  shows {0} ◁R unfolding Ideal_def
  using Ring_ZF_1_L6 add_group.unit_singl_subgr by auto
```

An ideal is s subset of the the ring.

```
lemma (in ring0) ideal_dest_subset:
  assumes I ◁R
  shows I ⊆ R using assms add_group.group0_3_L2
  unfolding Ideal_def by auto
```

Ideals are closed with respect to the ring addition.

```
lemma (in ring0) ideal_dest_sum:
  assumes I ◁R x∈I y∈I
  shows x+y ∈I using assms add_group.group0_3_L6
  unfolding Ideal_def by auto
```

Ideals are closed with respect to the ring multiplication.

```
lemma (in ring0) ideal_dest_mult:
  assumes I ◁R x∈I y∈R
  shows x·y ∈I y·x ∈I using assms unfolding Ideal_def by auto
```

Ideals are closed with respect to taking the opposite in the ring.

```
lemma (in ring0) ideal_dest_minus:
  assumes I ◁R x∈I
  shows (-x) ∈ I
  using assms add_group.group0_3_T3A unfolding Ideal_def by auto
```

Every ideals contains zero.

```
lemma (in ring0) ideal_dest_zero:
  assumes I ◁R
  shows 0 ∈ I
  using assms add_group.group0_3_L5 unfolding Ideal_def by auto
```

If the rules are satisfied, then we have an ideal

```
theorem (in ring0) ideal_intro:
  assumes ∀x∈I. ∀y∈I. x+y∈I
    ∀x∈I. ∀y∈R. x·y ∈I
    ∀x∈I. ∀y∈R. y·x ∈I
    I ⊆ R I≠0
  shows I◁R
proof -
  { fix x assume x∈I
    with assms(4) have (-x)∈R using  Ring_ZF_1_L3(1) by auto
    then have (-x) = 1·(-x) using Ring_ZF_1_L3(6) by auto
    with assms(4) ‹x∈I› have (-x) = -(1·x)
      using Ring_ZF_1_L7(2) Ring_ZF_1_L2(2) by auto
    with assms(4) ‹x∈I› have (-x) = (-1)·x
      using Ring_ZF_1_L7(1) Ring_ZF_1_L2(2) by auto
    with assms(3) ‹x∈I› have (-x) ∈I
      using Ring_ZF_1_L2(2) Ring_ZF_1_L3(1) by auto
  } hence ∀x∈I. GroupInv(R, A)(x) ∈ I by auto
  with assms(1,4,5) have IsAsubgroup(I,A)
    using add_group.group0_3_T3 unfolding IsOpClosed_def by auto
  moreover from assms(2,3) have ∀x∈I. ∀y∈R. y·x ∈ I ∧ x·y ∈ I by auto
  ultimately show thesis unfolding Ideal_def by auto
qed
```

The simplest way to obtain an ideal from others is the intersection, since the intersection of arbitrary collection of ideals is an ideal.

theorem (**in** ring0) intersection_ideals:
  assumes $\forall$ J$\in\mathcal{J}$. (J $\triangleleft$R) $\mathcal{J} \neq$ 0
  shows ($\bigcap\mathcal{J}$) $\triangleleft$R
  using assms ideal_dest_mult add_group.subgroup_inter
  unfolding Ideal_def **by** auto

In particular, intersection of two ideals is an ideal.

corollary (**in** ring0) inter_two_ideals: **assumes** I$\triangleleft$R J$\triangleleft$R
  shows (I$\cap$J) $\triangleleft$R
proof -
  let $\mathcal{J}$ = {I,J}
  from assms **have** $\forall$ J$\in\mathcal{J}$. (J $\triangleleft$R) **and** $\mathcal{J} \neq$ 0 **by** simp_all
  then have ($\bigcap\mathcal{J}$) $\triangleleft$R **using** intersection_ideals **by** blast
  thus thesis **by** simp
qed

From any set, we may construct the minimal ideal containing that set

definition (**in** ring0) generatedIdeal ($\langle$_$\rangle_I$)
  where X$\subseteq$R $\Longrightarrow$ $\langle$X$\rangle_I$ $\equiv$ $\bigcap${I$\in\mathcal{I}$. X $\subseteq$ I}

The ideal generated by a set is an ideal

corollary (**in** ring0) generated_ideal_is_ideal:
  assumes X$\subseteq$R **shows** $\langle$X$\rangle_I$ $\triangleleft$R
proof -
  let $\mathcal{J}$ = {I$\in\mathcal{I}$. X $\subseteq$ I}
  have $\forall$ J $\in$ $\mathcal{J}$. (J $\triangleleft$R) **by** auto
  with assms **have** ($\bigcap\mathcal{J}$) $\triangleleft$R **using** ring_self_ideal intersection_ideals
by blast
  with assms **show** thesis **using** generatedIdeal_def **by** simp
qed

The ideal generated by a set is contained in any ideal containing the set.

corollary (**in** ring0) generated_ideal_small:
  assumes X$\subseteq$I I $\triangleleft$R
  shows $\langle$X$\rangle_I$ $\subseteq$ I
proof -
  from assms **have** I$\in${J$\in$Pow(R). J $\triangleleft$R $\wedge$ X$\subseteq$J}
    using ideal_dest_subset **by** auto
  then have $\bigcap${J$\in$Pow(R). J $\triangleleft$R $\wedge$ X$\subseteq$J} $\subseteq$ I **by** auto
  moreover from assms **have** X$\subseteq$ R **using** ideal_dest_subset **by** auto
  ultimately show $\langle$X$\rangle_I$ $\subseteq$ I **using** generatedIdeal_def **by** auto
qed

The ideal generated by a set contains the set.

corollary (**in** ring0) generated_ideal_contains_set:
  assumes X$\subseteq$R **shows** X$\subseteq\langle$X$\rangle_I$
  using assms ring_self_ideal generatedIdeal_def **by** auto

To be able to show properties of an ideal generated by a set, we have the following induction result

```
lemma (in ring0) induction_generated_ideal:
  assumes
    X≠0
    X⊆R
    ∀y∈R. ∀z∈R. ∀q∈⟨X⟩I. P(q) ⟶ P(y·q·z)
    ∀y∈R. ∀z∈R. P(y) ∧ P(z) ⟶ P(y+z)
    ∀x∈X. P(x)
  shows ∀y∈⟨X⟩I. P(y)
proof -
  let J = {m∈⟨X⟩I. P(m)}
  from assms(2,5) have X⊆J
    using generated_ideal_contains_set by auto
  from assms(2) have J⊆R
    using generated_ideal_is_ideal ideal_dest_subset by auto
  moreover
  { fix y z assume y∈R z∈J
    then have y∈R 1∈R z∈⟨X⟩I P(z)
      using Ring_ZF_1_L2(2) by simp_all
    with assms(3) have P(y·z·1) and P(1·z·y) by simp_all
    with ‹J⊆R› ‹y∈R› ‹z∈J› have P(y·z) and P(z·y)
      using Ring_ZF_1_L4(3) Ring_ZF_1_L3(5,6) by auto
    with assms(2) ‹z∈⟨X⟩I› ‹y∈R› have y·z∈J z·y∈J
      using ideal_dest_mult generated_ideal_is_ideal
      by auto
  } hence ∀x∈J. ∀y∈R. y · x ∈ J ∧ x · y ∈ J by auto
  moreover have IsAsubgroup(J,A)
  proof -
    from assms(1) ‹X⊆J› ‹J⊆R› have J≠0 and J⊆R
      by auto
    moreover
    { fix x y assume as: x∈J y∈J
      with assms(2,4) have P(x+y)
        using ideal_dest_subset generated_ideal_is_ideal
        by blast
      with assms(2) ‹x∈J› ‹y∈J› have x+y ∈ J
        using generated_ideal_is_ideal ideal_dest_sum
        by auto
    } then have J {is closed under} A
      unfolding IsOpClosed_def by auto
    moreover
    { fix x assume x∈J
      with ‹J⊆R› have x∈⟨X⟩I x∈R P(x) by auto
      with assms(3) have P((-1)·x·1)
        using Ring_ZF_1_L2(2) Ring_ZF_1_L3(1) by simp
      with assms(2) ‹x∈⟨X⟩I› ‹x∈R› have (-x)∈J
        using Ring_ZF_1_L3(1,5,6) Ring_ZF_1_L7(1) Ring_ZF_1_L2(2)
          generated_ideal_is_ideal ideal_dest_minus by auto
```

```
    } hence ∀x∈J. (-x)∈J by simp
    ultimately show IsAsubgroup(J,A)
      by (rule add_group.group0_3_T3)
  qed
  ultimately have J◁R unfolding Ideal_def by auto
  with ‹X⊆J› show thesis using generated_ideal_small by auto
qed
```

An ideal is very particular with the elements it may contain. If it contains the neutral element of multiplication then it is in fact the whole ring and not a proper subset.

```
theorem (in ring0) ideal_with_one:
  assumes I◁R 1∈I shows I = R
  using assms ideal_dest_subset ideal_dest_mult(2) Ring_ZF_1_L3(5)
  by force
```

The only ideal containing an invertible element is the whole ring.

```
theorem (in ring0) ideal_with_unit:
  assumes I◁R x∈I ∃y∈R. y·x = 1 ∨ x·y =1
  shows I = R
  using assms ideal_with_one unfolding Ideal_def by blast
```

The previous result drives us to define what a maximal ideal would be: an ideal such that any bigger ideal is the whole ring:

```
definition (in ring0) maximalIdeal (_◁mR) where
  I◁mR ≡ I◁R ∧ I≠R ∧ (∀J∈𝓘. I⊆J ∧ J≠R ⟶ I=J)
```

Before delving into maximal ideals, lets define some operation on ideals that are useful when formulating some proofs. The product ideal of ideals I, J is the smallest ideal containing all products of elements from I and J:

```
definition (in ring0) productIdeal (infix ·I 90) where
  I◁R ⟹ J◁R  ⟹ I·IJ ≡ ⟨M(I×J)⟩I
```

The sum ideal of ideals is the smallest ideal containg both I and J:

```
definition (in ring0) sumIdeal (infix +I 90) where
  I◁R ⟹ J◁R  ⟹ I+IJ ≡ ⟨I∪J⟩I
```

Sometimes we may need to sum an arbitrary number of ideals, and not just two.

```
definition(in ring0) sumArbitraryIdeals (⊕I_ 90) where
  𝓙 ⊆ 𝓘 ⟹ ⊕I𝓙 ≡ ⟨⋃𝓙⟩I
```

Each component of the sum of ideals is contained in the sum.

```
lemma (in ring0) comp_in_sum_ideals:
  assumes I◁R and J◁R
  shows I ⊆ I+IJ and J ⊆ I+IJ and I∪J ⊆ I+IJ
proof -
```

```
  from assms have I∪J ⊆ R using ideal_dest_subset
    by auto
  with assms show I ⊆ I+IJ J ⊆ I+IJ I∪J ⊆ I+IJ
    using generated_ideal_contains_set sumIdeal_def
    by auto
qed
```

Every element in the arbitrary sum of ideals is generated by only a finite subset of those ideals

```
lemma (in ring0) sum_ideals_finite_sum:
  assumes 𝒥 ⊆ ℐ s∈(⊕I𝒥)
  shows ∃𝒯∈FinPow(𝒥). s∈(⊕I𝒯)
proof -
  { assume ⋃𝒥=0
    then have 𝒥⊆{0} by auto
    with assms(2) have thesis
      using subset_Finite nat_into_Finite
      unfolding FinPow_def by blast
  }
  moreover
  { let P = λt. ∃𝒯∈FinPow(𝒥). t∈(⊕I𝒯)
    assume ⋃𝒥≠0
    moreover from assms(1) have ⋃𝒥⊆R by auto
    moreover
    { fix y z q assume P(q) y∈R z∈R q∈⟨⋃𝒥⟩I
      then obtain 𝒯 where 𝒯∈FinPow(𝒥) and q ∈ ⊕I𝒯
        by auto
      from assms(1) ‹𝒯∈FinPow(𝒥)› have ⋃𝒯 ⊆ R 𝒯⊆ℐ
        unfolding FinPow_def by auto
      with ‹q∈⊕I𝒯› ‹y∈R› ‹z∈R› have y·q·z ∈ ⊕I𝒯
        using generated_ideal_is_ideal sumArbitraryIdeals_def
        unfolding Ideal_def by auto
      with ‹𝒯∈FinPow(𝒥)› have P(y·q ·z) by auto
    } hence ∀y∈R. ∀z∈R. ∀q∈⟨⋃𝒥⟩I. P(q) ⟶ P(y·q·z)
      by auto
    moreover
    { fix y z assume P(y) P(z)
      then obtain Ty Tz where T: Ty∈FinPow(𝒥) y ∈ ⊕ITy
        Tz∈FinPow(𝒥) z ∈ ⊕ITz by auto
      from T(1,3) have A: Ty∪Tz ∈ FinPow(𝒥)
        unfolding FinPow_def using Finite_Un by auto
      with assms(1) have ⋃Ty ⊆ ⋃(Ty∪Tz) ⋃Tz ⊆ ⋃(Ty∪Tz) and
        sub: ⋃(Ty∪Tz) ⊆ R
        unfolding FinPow_def by auto
      then have ⋃(Ty∪Tz) ⊆ ⟨⋃(Ty∪Tz)⟩I
        using generated_ideal_contains_set by simp
      hence ⋃Ty ⊆ ⟨⋃(Ty∪Tz)⟩I ⋃Tz ⊆ ⟨⋃(Ty∪Tz)⟩I by auto
      with sub have ⟨⋃Ty⟩I⊆ ⟨⋃(Ty∪Tz)⟩I ⟨⋃Tz⟩I ⊆ ⟨⋃(Ty∪Tz)⟩I
        using generated_ideal_small generated_ideal_is_ideal
```

```
        by auto
      moreover from assms(1) T(1,3) have T_y⊆𝓘 T_z⊆𝓘
        unfolding FinPow_def by auto
      moreover note T(2,4)
      ultimately have y ∈ ⟨⋃(T_y∪T_z)⟩_I z ∈ ⟨⋃(T_y∪T_z)⟩_I
        using sumArbitraryIdeals_def sumArbitraryIdeals_def
        by auto
      with ‹⋃(T_y∪T_z) ⊆ R› have y+z ∈ ⟨⋃(T_y∪T_z)⟩_I
        using generated_ideal_is_ideal ideal_dest_sum by auto
      moreover
      from ‹T_y⊆𝓘› ‹T_z⊆𝓘› have T_y∪T_z ⊆ 𝓘 by auto
      then have (⊕_I(T_y∪T_z)) = ⟨⋃(T_y∪T_z)⟩_I
        using sumArbitraryIdeals_def by auto
      ultimately have y+z ∈ (⊕_I(T_y∪T_z)) by simp
      with A have P(y+z) by auto
    } hence ∀y∈R. ∀z∈R. P(y) ∧ P(z) ⟶ P(y + z)
      by auto
    moreover
    { fix t assume t∈⋃𝒥
      then obtain J where t∈J J∈𝒥 by auto
      then have {J}∈FinPow(𝒥) unfolding FinPow_def
        using eqpoll_imp_Finite_iff nat_into_Finite
        by auto
      moreover
      from assms(1) ‹J∈𝒥› have (⊕_I{J}) = ⟨J⟩_I
        using sumArbitraryIdeals_def by auto
      with assms(1) ‹t∈J› ‹J∈𝒥› have t∈(⊕_I{J})
        using generated_ideal_contains_set by force
      ultimately have P(t) by auto
    } hence ∀t∈⋃𝒥. P(t) by auto
    ultimately have ∀t∈⟨⋃𝒥⟩_I. P(t)
      by (rule induction_generated_ideal)
    with assms have thesis using sumArbitraryIdeals_def
      by auto
  }
  ultimately show thesis by auto
qed
```

By definition of product of ideals and of an ideal itself, it follows that the product of ideals is an ideal contained in the intersection

```
theorem (in ring0) product_in_intersection:
  assumes I◁R J◁R
  shows I·_I J ⊆ I∩J and (I·_I J)◁R and M(I×J) ⊆ I·_I J
proof -
  have M(I×J) ⊆ I∩J
  proof
    fix x assume x∈M(I×J)
    then obtain y where y∈I×J ⟨y,x⟩∈M unfolding image_def
      by auto
```

```
    then obtain yᵢ yⱼ where y: yᵢ∈I yⱼ∈J ⟨⟨yᵢ,yⱼ⟩,x⟩ ∈ M
      by auto
    from assms have I⊆R J⊆R using ideal_dest_subset by simp_all
    with ringAssum assms y show x∈I∩J
      unfolding Ideal_def IsAring_def IsAmonoid_def IsAssociative_def
      using apply_equality by force
  qed
  with assms show I·ᵢJ ⊆ I∩J
    using productIdeal_def generated_ideal_small inter_two_ideals
    by auto
  from assms ‹M(I×J) ⊆ I∩J› have M(I×J) ⊆ R
    using ideal_dest_subset by auto
  with assms show (I·ᵢJ) ◁R and M(I×J) ⊆ I·ᵢJ
    using productIdeal_def generated_ideal_is_ideal
      generated_ideal_contains_set by auto
qed
```

We will show now that the sum of ideals is no more that the sum of the ideal elements.

```
lemma (in ring0) sum_elements:
  assumes I ◁R J ◁R x∈I y∈J
  shows x+y ∈ I+ᵢJ
proof -
  from assms(1,2) have I∪J ⊆ R using ideal_dest_subset
    by auto
  with assms(3,4) have x ∈ ⟨I∪J⟩ᵢ y ∈ ⟨I∪J⟩ᵢ
    using generated_ideal_contains_set by auto
  with assms(1,2) ‹I∪J ⊆ R› show thesis
    using generated_ideal_is_ideal ideal_dest_subset ideal_dest_sum
      sumIdeal_def by auto
qed
```

For two ideals the set containing all sums of their elements is also an ideal.

```
lemma (in ring0) sum_elements_is_ideal:
  assumes I ◁R J ◁R
  shows (A(I×J)) ◁R
proof -
  from assms have ij: I×J ⊆ R×R using ideal_dest_subset by auto
  have Aim: A(I×J) ⊆ R using add_group.group_oper_fun func1_1_L6(2)
    by auto
  moreover
  { fix x y assume x∈R y ∈ A(I×J)
    from ij ‹y∈A(I×J)› obtain yᵢ yⱼ where
      y: y=yᵢ+yⱼ yᵢ∈I yⱼ∈J
      using add_group.group_oper_fun func_imagedef by auto
    from ‹x∈R› y ij have
      x·y = (x·yᵢ)+(x·yⱼ) and y·x = (yᵢ·x)+(yⱼ·x)
      using ring_oper_distr add_group.group_op_closed by auto
    moreover from assms ‹x∈R› y(2,3) have
```

```
    x·y_i ∈ I y_i·x ∈ I x·y_j ∈ J y_j·x ∈ J
    using  ideal_dest_mult by auto
  ultimately have x·y∈A(I×J) y·x∈A(I×J)
    using ij add_group.group_oper_fun func_imagedef by auto
} hence ∀x∈A(I×J). ∀y∈R. y · x ∈ A(I×J) ∧ x · y ∈ A(I×J)
  by auto
moreover have IsAsubgroup(A(I×J),A)
proof -
  from assms ij have 0+0 ∈ A  (I × J)
    using add_group.group_oper_fun ideal_dest_zero func_imagedef
    by auto
  with Aim have A(I × J) ≠ 0 and A(I × J) ⊆ R
    by auto
  moreover
  { fix x y assume xy: x ∈ A(I × J) y ∈ A(I × J)
    with ij obtain x_i x_j y_i y_j where
      x_i∈I x_j∈J x=x_i+x_j y_i∈I y_j∈J y=y_i+y_j
      using add_group.group_oper_fun func_imagedef by auto
    from ‹x∈A(I × J)› ‹A(I × J)⊆R› have x∈R by auto
    from assms ‹x_i∈I› ‹x_j∈J› ‹y_i∈I› ‹y_j∈J›
      have x_i∈R x_j∈R y_i∈R y_j∈R
        using ideal_dest_subset by auto
    from ij ‹x∈R› ‹y_i∈R› ‹y_j∈R› ‹y=y_i+y_j›
      have x+y = (x+y_i)+y_j
      using Ring_ZF_1_L10(1) by simp
    with ‹x=x_i+x_j› ‹x_i∈R› ‹x_j∈R› ‹y_i∈R› ‹y_j∈R›
      have x+y = (x_i+y_i)+(x_j+y_j)
      using Ring_ZF_2_L5(4) by simp
    moreover from assms ‹x_i∈I› ‹x_j∈J› ‹y_i∈I› ‹y_j∈J›
      have x_i+y_i ∈ I and x_j+y_j ∈ J
      using ideal_dest_sum by auto
    ultimately have x+y ∈ A(I×J)
      using ij add_group.group_oper_fun func_imagedef
      by auto
  } then have A(I × J) {is closed under} A
    unfolding IsOpClosed_def by auto
  moreover
  { fix x assume x ∈ A(I × J)
    with ij obtain x_i x_j where x_i∈I x_j∈J x=x_i+x_j
      using add_group.group_oper_fun func_imagedef by auto
    with assms have (-x) = (-x_i)-x_j
      using Ring_ZF_1_L9(2) ideal_dest_subset by auto
    moreover from assms ‹x_i∈I› ‹x_j∈J›
    have (-x_i)∈I and (-x_j)∈J
      using ideal_dest_minus by simp_all
    ultimately have (-x) ∈ A(I×J)
      using add_group.group_oper_fun ij func_imagedef by auto
  } hence ∀x∈A(I × J). (- x) ∈ A(I × J) by auto
  ultimately show thesis using add_group.group0_3_T3
```

```
    by simp
  qed
  ultimately show (A(I × J)) ◁R unfolding Ideal_def by auto
qed
```

The set of all sums of elements of two ideals is their sum ideal i.e. the ideal generated by their union.

```
corollary (in ring0) sum_ideals_is_sum_elements:
  assumes  I ◁R J ◁R
  shows (A(I × J)) = I+_I J
proof
  from assms have ij: I⊆R J⊆R using ideal_dest_subset
    by auto
  then have ij_prd: I×J ⊆ R×R by auto
  with assms show A(I × J) ⊆ I +_I J
    using add_group.group_oper_fun sum_elements func_imagedef
    by auto
  { fix x assume x∈I
    with ij(1) have x=x+0 using Ring_ZF_1_L3(3) by auto
    with assms(2) ij_prd ‹x∈I› have x∈A  (I × J)
      using add_group.group0_3_L5 add_group.group_oper_fun func_imagedef
      unfolding Ideal_def by auto
  } hence I ⊆ A(I × J) by auto
  moreover
  { fix x assume x: x∈J
    with ij(2) have x=0+x using Ring_ZF_1_L3(4) by auto
    with assms(1) ij_prd ‹x∈J› have x ∈ A(I × J)
      using add_group.group0_3_L5 add_group.group_oper_fun func_imagedef
      unfolding Ideal_def by auto
  } hence J ⊆ A(I × J) by auto
  ultimately have I∪J ⊆ A(I × J) by auto
  with assms show I+_I J ⊆ A(I × J)
    using generated_ideal_small sum_elements_is_ideal sumIdeal_def
    by auto
qed
```

The sum ideal of two ideals is indeed an ideal.

```
corollary (in ring0) sum_ideals_is_ideal:
  assumes I ◁R J ◁R
  shows (I+_I J) ◁R using assms sum_ideals_is_sum_elements
  sum_elements_is_ideal ideal_dest_subset by auto
```

The operation of taking the sum of ideals is commutative.

```
corollary (in ring0) sum_ideals_commute:
  assumes I◁R J◁R
  shows (I +_I J) = (J +_I I)
proof -
  have I ∪ J = J ∪ I by auto
  with assms show thesis using sumIdeal_def by auto
```

qed

Now that we know what the product of ideals is, we are able to define what a prime ideal is:

```
definition (in ring0) primeIdeal (_◁pR) where
  P◁pR ≡ P◁R ∧ P≠R ∧ (∀I∈ℐ. ∀J∈ℐ. I·IJ ⊆ P ⟶ (I⊆P ∨ J⊆P))
```

Any maximal ideal is a prime ideal.

```
theorem (in ring0) maximal_is_prime:
  assumes Q◁mR shows Q◁pR
proof -
  have Q∈ℐ using assms ideal_dest_subset
    unfolding maximalIdeal_def by auto
  { fix I J assume I∈ℐ J∈ℐ I·IJ ⊆ Q
    { assume K: ¬(I⊆Q) ¬(J⊆Q)
      from ‹¬(I⊆Q)› obtain x where x∈I-Q by auto
      with ‹I∈ℐ› have x∈R using ideal_dest_subset by auto
      from ‹I∈ℐ› ‹J∈ℐ› have sub: I×J ⊆ R×R by auto
      let K = ⟨Q∪{x}⟩I
      from ‹Q∈ℐ› ‹x∈R› have Q∪{x} ⊆ R by auto
      then have Q⊆K and x∈K using generated_ideal_contains_set
        by auto
      with ‹x∈I-Q› have Q⊆K K≠Q by auto
      from ‹Q∪{x} ⊆ R› have K∈ℐ
        using ideal_dest_subset generated_ideal_is_ideal by blast
      with assms ‹Q⊆K› ‹K≠Q› have K=R unfolding maximalIdeal_def
        by auto
      let P = Q+II
      from ‹Q∈ℐ› ‹I∈ℐ› ‹x∈I-Q› have Q∪{x} ⊆ Q+II
        using comp_in_sum_ideals(3) by auto
      with ‹Q∈ℐ› ‹I∈ℐ› have K ⊆ Q+II Q+II ⊆ R
        using sum_ideals_is_ideal generated_ideal_small ideal_dest_subset
        by simp_all
      with ‹K=R› have 1∈Q+II  using Ring_ZF_1_L2(2) by auto
      with ‹I∈ℐ› ‹Q∈ℐ› have 1∈A(Q×I)
        using sum_ideals_is_sum_elements by auto
      moreover from ‹I∈ℐ› ‹Q∈ℐ› have Q×I ⊆ R×R by auto
      ultimately obtain xm xi where xm∈Q xi∈I 1=xm+xi
        using func_imagedef add_group.group_oper_fun by auto
      { fix y assume y∈J
        with ‹xm∈Q› ‹xi∈I› ‹Q∈ℐ› ‹I∈ℐ› ‹J∈ℐ›
        have y∈R xm∈R xi∈R by auto
        with ‹y∈J› ‹J∈ℐ› ‹1=xm+xi› have (xm·y)+(xi·y) = y
          using Ring_ZF_1_L3(6) ring_oper_distr(2) by simp
        from ‹Q∈ℐ› ‹xm∈Q›  ‹y∈R› have xm·y∈Q
          using ideal_dest_mult(1) by auto
        from sub ‹y∈J› ‹xi∈I› ‹I∈ℐ› ‹J∈ℐ› ‹I·IJ⊆Q› have xi·y∈Q
          using func_imagedef mult_monoid.monoid_oper_fun
            product_in_intersection(3) by force
```

```
      with ‹Q∈ℐ› ‹(xₘ·y)+(xᵢ·y) = y› ‹xₘ·y∈Q› have y∈Q
        using ideal_dest_sum by force
    } hence J ⊆ Q by auto
    with K have False by auto
  } hence (I⊆Q)∨(J⊆Q) by auto
} hence ∀I∈ℐ. ∀J∈ℐ. I ·_I J ⊆ Q ⟶ (I ⊆ Q ∨ J ⊆ Q) by auto
with assms show thesis unfolding maximalIdeal_def primeIdeal_def
  by auto
qed
```

In case of non-commutative rings, the zero divisor concept is too constrictive. For that we define the following concept of a prime ring. Note that in case that our ring is commutative, this is equivalent to having no zero divisors (there is no of that proof yet).

```
definition primeRing ([_,_,_]{is a prime ring}) where
  IsAring(R,A,M) ⟹ [R,A,M]{is a prime ring} ≡
    (∀x∈R. ∀y∈R. (∀z∈R. M⟨M⟨x,z⟩,y⟩ = TheNeutralElement(R,A)) ⟶
    x=TheNeutralElement(R,A) ∨ y=TheNeutralElement(R,A))
```

Prime rings appear when the zero ideal is prime.

```
lemma (in ring0) prime_ring_zero_prime_ideal:
  assumes [R,A,M]{is a prime ring} R≠{0}
  shows {0} ◃_p R
proof -
  {
    fix I J assume ij: I∈ℐ J∈ℐ I·_I J ⊆ {0}
    from ij(1,2) have I×J ⊆ R×R by auto
    { assume ¬(I⊆{0}) ¬(J⊆{0})
      then obtain x_i x_j where x_i≠0 x_j≠0 x_i∈I x_j∈J
        by auto
      from ij(1,2) ‹x_i∈I› ‹x_j∈J› have x_i∈R x_j∈R by auto
      { fix u assume u∈R
        with ‹I∈ℐ› ‹x_i∈I› ‹x_j∈J› ‹I×J ⊆ R×R›
          have x_i·u·x_j ∈ M(I×J)
          using ideal_dest_mult(1) func_imagedef
            mult_monoid.monoid_oper_fun by auto
        with ij have x_i·u·x_j = 0
          using product_in_intersection(3) by blast
      } hence ∀u∈R. x_i·u·x_j = 0 by simp
      with ringAssum assms(1) ‹x_i∈R› ‹x_j∈R› ‹x_i≠0› ‹x_j≠0›
        have False using primeRing_def by auto
    } hence I⊆{0} ∨ J⊆{0} by auto
  } hence ∀I∈ℐ. ∀J∈ℐ. I ·_I J ⊆ {0} ⟶ (I ⊆ {0} ∨ J ⊆ {0})
    by auto
  with assms(2) show thesis
    using zero_ideal unfolding primeIdeal_def by blast
qed
```

If the trivial ideal $\{0\}$ is a prime ideal then the ring is a prime ring.

```
lemma (in ring0) zero_prime_ideal_prime_ring:
  assumes {0}◁pR
  shows [R,A,M]{is a prime ring}
proof -
  { fix x y z assume x∈R y∈R ∀z∈R. x·z·y = 0
    let X = ⟨{x}⟩I
    let Y = ⟨{y}⟩I
    from ‹x∈R› ‹y∈R› have X×Y ⊆ R×R
      using generated_ideal_is_ideal ideal_dest_subset
      by auto
    let P = λq. (∀z∈Y. q·z = 0)
    let Q = λq. (∀z∈R. x·z·q =0)
    have ∀y∈Y. Q(y)
    proof -
      from ‹y∈R› have {y}≠0 and {y}⊆R by auto
      moreover
      { fix s t q assume yzq: s∈R t∈R q ∈ ⟨{y}⟩I ∀k∈R. x·k·q = 0
        from ‹y∈R› yzq(3) have q∈R
          using generated_ideal_is_ideal ideal_dest_subset by auto
        { fix u assume u∈R
          from yzq(1,2) ‹x∈R› ‹q∈R› ‹u∈R›
            have x·u·(s·q·t) = (x·(u·s)·q)·t
            using Ring_ZF_1_L11(2) Ring_ZF_1_L4(3) by auto
          with ‹u∈R› yzq(1,2,4) have x·u·(s·q·t) = 0
            using Ring_ZF_1_L4(3) Ring_ZF_1_L6(1) by simp
        } hence ∀za∈R. x·za·(s·q·t) = 0 by simp
      } hence ∀t∈R. ∀z∈R. ∀q∈⟨{y}⟩I.
        (∀z∈R. x·z·q = 0) ⟶ (∀za∈R. x·za·(t·q·z) = 0) by simp
      moreover from ‹x∈R› have ∀y∈R. ∀z∈R.
            (∀z∈R. x·z·y = 0) ∧ (∀za∈R. x·za·z = 0) ⟶
            (∀za∈R. x·za·(y+z) = 0)
        using ring_oper_distr(1) Ring_ZF_1_L4(3) Ring_ZF_1_L3(3)
          Ring_ZF_1_L2(1) by simp
      moreover from ‹∀z∈R. x·z·y = 0› have ∀xa∈{y}. ∀z∈R. x·z·xa = 0
by simp
      ultimately show thesis by (rule induction_generated_ideal)
    qed
    from ‹x∈R› ‹∀y∈Y. Q(y)› have ∀y∈Y. x·y = 0
      using Ring_ZF_1_L2(2) Ring_ZF_1_L3(5) by force
    from ‹y∈R› have Y◁R Y⊆R
      using generated_ideal_is_ideal ideal_dest_subset by auto
    have ∀y∈X. P(y)
    proof -
      from ‹x∈R› have {x}≠0 {x}⊆R by auto
      moreover
      { fix q1 q2 q3
        assume q: q1∈R q2∈R q3 ∈ ⟨{x}⟩I ∀k∈Y. q3·k = 0
        from ‹x∈R› ‹q3 ∈ ⟨{x}⟩I› have q3∈R
          using generated_ideal_is_ideal ideal_dest_subset by auto
```

```
        with ‹Y◁R› ‹Y⊆R› q(1,2,4) have ∀zₐ∈⟨{y}⟩_I. q₁·q₃·q₂ · zₐ = 0
          using ideal_dest_mult(2) Ring_ZF_1_L4(3) Ring_ZF_1_L11(2)
            Ring_ZF_1_L6(2) by auto
      } hence ∀yₐ∈R. ∀z∈R. ∀q∈⟨{x}⟩_I.
              (∀z∈⟨{y}⟩_I. q·z = 0) ⟶ (∀zₐ∈⟨{y}⟩_I. yₐ·q·z·zₐ = 0)
        by auto
      moreover from ‹Y⊆R› have ∀yₐ∈R. ∀z∈R.
        (∀z∈⟨{y}⟩_I. yₐ·z = 0) ∧ (∀zₐ∈⟨{y}⟩_I. z·zₐ = 0) ⟶
          (∀zₐ∈⟨{y}⟩_I. (yₐ+z)·zₐ = 0)
        using ring_oper_distr(2) Ring_ZF_1_L3(3) Ring_ZF_1_L2(1)
        by auto
      moreover from ‹∀y∈Y. x·y = 0› have ∀x∈{x}. ∀z∈⟨{y}⟩_I. x·z = 0
by auto
      ultimately show thesis by (rule induction_generated_ideal)
    qed
    from ‹X×Y ⊆ R×R› ‹∀y∈X. P(y)› have M(X×Y) ⊆ {0}
      using mult_monoid.monoid_oper_fun func_imagedef by auto
    with ‹x∈R› ‹y∈R› have X·_IY ⊆ {0}
      using generated_ideal_small zero_ideal productIdeal_def
        generated_ideal_is_ideal by auto
    with assms ‹x∈R› ‹y∈R› have X ⊆ {0} ∨ Y ⊆ {0}
      using generated_ideal_is_ideal ideal_dest_subset
      unfolding primeIdeal_def by auto
    with ‹x∈R› ‹y∈R› have x=0 ∨ y=0
      using generated_ideal_contains_set by auto
  } hence ∀x∈R. ∀y∈R. (∀z∈R. x·z·y = 0) ⟶ x=0 ∨ y=0
    by auto
  with ringAssum show thesis using primeRing_def by auto
qed
```

We can actually use this definition of a prime ring as a condition to check for prime ideals.

```
theorem (in ring0) equivalent_prime_ideal:
  assumes P◁_pR
  shows ∀x∈R. ∀y∈R. (∀z∈R. x·z·y∈P) ⟶ x∈P ∨ y∈P
proof -
  { fix x y assume x∈R y∈R ∀z∈R. x·z·y∈P y∉P
    let X = ⟨{x}⟩_I
    let Y = ⟨{y}⟩_I
    from ‹y∈R› have {y}≠0 and {y}⊆R by auto
    moreover have ∀yₐ∈R.
       ∀z∈R. ∀q∈⟨{y}⟩_I. (∀t∈⟨{x}⟩_I. ∀u∈R. t·u·q ∈ P) ⟶
                (∀t∈⟨{x}⟩_I. ∀u∈R. t·u·(yₐ·q·z) ∈ P)
    proof -
      { fix yₐ z q t u
        assume yₐ∈R z∈R q∈Y ∀t∈X. ∀u∈R. t·u·q∈P t∈X u∈R
        from ‹x∈R› ‹y∈R› ‹q∈Y› ‹t∈X› have q∈R t∈R
          using generated_ideal_is_ideal ideal_dest_subset by auto
        from ‹yₐ∈R› ‹u∈R› have u·yₐ∈R using Ring_ZF_1_L4(3) by auto
```

```
      with assms ‹z∈R› ‹∀t∈X. ∀u∈R. t·u·q∈P› ‹t∈X›
        have (t·(u·ya)·q)·z ∈ P
        using ideal_dest_mult(1) unfolding primeIdeal_def by auto
      with ‹ya∈R› ‹z∈R› ‹u∈R› ‹q∈R› ‹t∈R› have t·u·(ya·q·z) ∈ P
        using Ring_ZF_1_L4(3) Ring_ZF_1_L11(2) by auto
    } thus thesis by simp
  qed
  moreover have ∀y∈R. ∀z∈R.
    (∀t∈⟨{x}⟩I. ∀u∈R. t·u·y ∈ P) ∧ (∀t∈⟨{x}⟩I. ∀u∈R. t·u·z ∈ P) ⟶
    (∀t∈⟨{x}⟩I. ∀u∈R. t·u·(y+z) ∈ P)
  proof -
    { fix ya z t u assume ass: ya∈R z∈R t∈X u∈R
        ∀t∈⟨{x}⟩I. ∀u∈R. t·u·ya∈P ∀t∈⟨{x}⟩I. ∀u∈R. t·u·z∈P
      from ‹x∈R› ‹t∈X› have t∈R
        using ideal_dest_subset generated_ideal_is_ideal by auto
      from assms ass(3,4,5,6) have (t·u·ya)+(t·u·z) ∈ P
        using ideal_dest_sum unfolding primeIdeal_def by simp
      with ‹t∈R› ‹ya∈R› ‹z∈R› ‹u∈R› have t·u·(ya+z) ∈ P
        using Ring_ZF_1_L4(3) ring_oper_distr(1) by simp
    } thus thesis by simp
  qed
  moreover have ∀ya∈⟨{x}⟩I. ∀u∈R. ya·u·y ∈ P
  proof -
    from ‹x∈R› have {x}≠0 and {x}⊆R by auto
    moreover have ∀ya∈R. ∀z∈R. ∀q∈X.
      (∀u∈R. q·u·y ∈ P) ⟶ (∀u∈R. ya·q·z·u·y ∈ P)
    proof -
      { fix ya z q u assume
          ass: ya∈R z∈R q∈X ∀u∈R. q·u·y ∈ P u ∈ R
        from ‹x∈R› ‹q∈X› have q: q∈R
          using generated_ideal_is_ideal ideal_dest_subset
          by auto
        from assms ass(1,2,4,5) have ya·(q·(z·u)·y) ∈ P
          using Ring_ZF_1_L4(3) ideal_dest_mult(2)
          unfolding primeIdeal_def by simp
        with ‹y∈R› ‹ya∈R› ‹z∈R› ‹u ∈ R› ‹q∈R›
          have ya·q·z·u·y ∈ P
          using Ring_ZF_1_L4(3) Ring_ZF_1_L11(2) by auto
      } thus thesis by auto
    qed
    moreover have ∀ya∈R. ∀z∈R.
      (∀u∈R. ya·u·y ∈ P) ∧ (∀u∈R. z·u·y ∈ P) ⟶ (∀u∈R. (ya+z)·u·y ∈
P)
    proof -
      { fix ya z u assume ya∈R z∈R u∈R
          ∀u∈R. ya·u·y ∈ P ∀u∈R. z·u·y ∈ P
        with assms ‹y∈R› have ((ya+z)·u)·y∈P
          using ideal_dest_sum ring_oper_distr(2) Ring_ZF_1_L4(3)
          unfolding primeIdeal_def by simp
```

```
      } thus thesis by simp
    qed
    moreover from ‹∀z∈R. x·z·y∈P› have ∀x∈{x}. ∀u∈R. x·u·y ∈ P
      by simp
    ultimately show thesis by (rule induction_generated_ideal)
  qed
  hence ∀xa∈{y}. ∀t∈⟨{x}⟩I. ∀u∈R. t·u·xa ∈ P by auto
  ultimately have R: ∀q∈Y. ∀t∈X. ∀u∈R. t·u·q ∈ P
    by (rule induction_generated_ideal)
  from ‹x∈R› ‹y∈R› have subprd: X×Y ⊆ R×R
    using ideal_dest_subset generated_ideal_is_ideal by auto
  { fix t assume t ∈ M(X×Y)
    with subprd obtain tx ty where
      tx∈X ty∈Y and t: t= tx·ty
      using func_imagedef mult_monoid.monoid_oper_fun
      by auto
    with R ‹tx∈X› ‹ty∈Y› have tx·1·ty ∈P
      using Ring_ZF_1_L2(2) by auto
    moreover from ‹x∈R› ‹tx∈X› have tx∈R
      using generated_ideal_is_ideal ideal_dest_subset by auto
    ultimately have t∈P using Ring_ZF_1_L3(5) t by auto
  } hence M(X×Y) ⊆ P by auto
  with assms ‹x∈R› ‹y∈R› have X ⊆ P ∨ Y ⊆ P
    unfolding primeIdeal_def
    using generated_ideal_small productIdeal_def
      generated_ideal_is_ideal ideal_dest_subset by auto
  with ‹x∈R› ‹y∈R› ‹y∉P› have x∈P using generated_ideal_contains_set

    by auto
} thus thesis by auto
qed
```

The next theorem provides a sufficient condition for a proper ideal P to be a prime ideal: if for all $x, y \in R$ it holds that for all $z \in R$ $xzy \in P$ only when $x \in P$ or $y \in P$ then P is a prime ideal.

```
theorem (in ring0) equivalent_prime_ideal_2:
  assumes ∀x∈R. ∀y∈R. (∀z∈R. x·z·y∈P) ⟶ x∈P ∨ y∈P P◁R P≠R
  shows  P◁pR
proof -
  { fix I J x xa
    assume I◁R I⊆R J◁R J⊆R I·IJ ⊆ P x∈J x∉P xa∈I
    from ‹I⊆R› ‹J⊆R› have I×J ⊆ R×R by auto
    { fix z assume z∈R
      with ‹I◁R› ‹x∈J› ‹xa∈I› ‹I×J ⊆ R×R› have (xa·z·x)∈M(I×J)
        using ideal_dest_mult(1) func_imagedef mult_monoid.monoid_oper_fun

        by auto
      with ‹I◁R› ‹J◁R› ‹I·IJ ⊆ P› have xa·z·x ∈ P
        using generated_ideal_contains_set func1_1_L6(2)
```

```
        mult_monoid.monoid_oper_fun productIdeal_def by force
    } with assms(1) ‹I⊆R› ‹J⊆R› ‹x∈J› ‹x∉P› ‹xₐ∈I› have xₐ∈P
      by blast
  } with assms(2,3) show thesis unfolding primeIdeal_def
    by auto
qed
```

48.2 Ring quotient

Similar to groups, rings can be quotiented by normal additive subgroups; but to keep the structure of the multiplicative monoid we need extra structure in the normal subgroup. This extra structure is given by the ideal.

Any ideal is a normal subgroup.

```
lemma (in ring0) ideal_normal_add_subgroup:
  assumes I◁R
  shows IsAnormalSubgroup(R,A,I)
  using ringAssum assms Group_ZF_2_4_L6(1)
  unfolding IsAring_def Ideal_def by auto
```

Each ring R is a group with respect to its addition operation. By the lemma `ideal_normal_add_subgroup` above an ideal $I \subseteq R$ is a normal subgroup of that group. Therefore we can define the quotient of the ring R by the ideal I using the notion of quotient of a group by its normal subgroup, see section `Normal subgroups and quotient groups` in `Group_ZF_2` theory.

```
definition (in ring0) QuotientBy where
  I◁R ⟹ QuotientBy(I) ≡ R//QuotientGroupRel(R,A,I)
```

Any ideal gives rise to an equivalence relation

```
corollary (in ring0) ideal_equiv_rel:
  assumes I◁R
  shows equiv(R,QuotientGroupRel(R,A,I))
  using assms add_group.Group_ZF_2_4_L3 unfolding Ideal_def
  by auto
```

Any quotient by an ideal is an abelian group.

```
lemma (in ring0) quotientBy_add_group:
  assumes I◁R
  shows IsAgroup(QuotientBy(I), QuotientGroupOp(R, A, I)) and
    QuotientGroupOp(R, A, I) {is commutative on} QuotientBy(I)
  using assms ringAssum ideal_normal_add_subgroup Group_ZF_2_4_T1
    Group_ZF_2_4_L6(2) QuotientBy_def QuotientBy_def Ideal_def
  unfolding IsAring_def by auto
```

Since every ideal is a normal subgroup of the additive group of the ring it is quite obvious that that addition is congruent with respect to the quotient group relation. The next lemma shows something a little bit less obvious:

that the multiplicative ring operation is also congruent with the quotient relation and gives rise to a monoid in the quotient.

```
lemma (in ring0) quotientBy_mul_monoid:
  assumes I◁R
  shows Congruent2(QuotientGroupRel(R, A, I),M) and
    IsAmonoid(QuotientBy(I),ProjFun2(R, QuotientGroupRel(R,A,I), M))
proof -
  let r = QuotientGroupRel(R,A,I)
  { fix x y s t assume ⟨x,y⟩ ∈ r and ⟨s,t⟩ ∈ r
    then have xyst: x∈R y∈R s∈R t∈R x-y ∈I s-t ∈I
      unfolding QuotientGroupRel_def by auto
    from ‹x∈R› ‹y∈R› ‹s∈R› have
      (x·s)-(y·t) = ((x·s)+((-(y·s))+(y·s)))-(y·t)
      using Ring_ZF_1_L3(1,3,7) Ring_ZF_1_L4(3,4) by simp
    with ‹x∈R› ‹y∈R› ‹s∈R› ‹t∈R› have
      (x·s)-(y·t) = ((x·s)-(y·s))+((y·s)-(y·t))
      using Ring_ZF_1_L3(1) Ring_ZF_1_L4(1,2,3) Ring_ZF_1_L10(1)
      by simp
    with ‹x∈R› ‹y∈R› ‹s∈R› ‹t∈R› have
      (x·s)-(y·t) = ((x-y)·s)+(y·(s-t))
      using Ring_ZF_1_L8 by simp
    with assms xyst have ⟨M⟨x,s⟩,M⟨y,t⟩⟩ ∈ r
      using ideal_dest_sum ideal_dest_mult(1,2) Ring_ZF_1_L4(3)
      unfolding QuotientGroupRel_def by auto
  } then show Congruent2(r,M) unfolding Congruent2_def by simp
  with assms show IsAmonoid(QuotientBy(I),ProjFun2(R,r, M))
    using add_group.Group_ZF_2_4_L3 mult_monoid.Group_ZF_2_2_T1 QuotientBy_def

    unfolding Ideal_def by auto
qed
```

Each ideal defines an equivalence relation on the ring with which both addition and multiplication are congruent. The next couple of definitions set up notation for the operations that result from projecting the ring addition and multiplication on the quotient space. We will write $x +_I y$ to denote the result of the quotient operation (with respect to an ideal I) on classes x and y

```
definition (in ring0) ideal_radd (_{+_}_) where
  x{+I}y ≡ QuotientGroupOp(R, A, I)⟨x,y⟩
```

Similarly $x \cdot_I y$ is the value of the projection of the ring's multiplication on the quotient space defined by the an ideal I, which as we know is a normal subgroup of the ring with addition.

```
definition (in ring0) ideal_rmult (_{·_}_) where
  x{·I}y ≡ ProjFun2(R, QuotientGroupRel(R,A,I), M)⟨x,y⟩
```

The value of the projection of taking the negative in the ring on the quotient space defined by an ideal I will be denoted `{-I}`.

```
definition (in ring0) ideal_rmin ({-_}_) where
  {-I}y ≡ GroupInv(QuotientBy(I),QuotientGroupOp(R, A, I))(y)
```

Subtraction in the quotient space is defined by the +I and -I operations in the obvious way.

```
definition (in ring0) ideal_rsub (_{-_}_) where
  x{-I}y ≡ x{+I}({-I}y)
```

The class of the zero of the ring with respect to the equivalence relation defined by an ideal I will be denoted 0_I.

```
definition (in ring0) ideal_rzero (0_) where
  0_I ≡ QuotientGroupRel(R,A,I){0}
```

Similarly the class of the neutral element of multiplication in the ring with respect to the equivalence relation defined by an ideal I will be denoted 1_I.

```
definition (in ring0) ideal_rone (1_) where
  1_I ≡ QuotientGroupRel(R,A,I){1}
```

The class of the sum of two units of the ring will be denoted 2_I.

```
definition (in ring0) ideal_rtwo (2_) where
  2_I ≡ QuotientGroupRel(R,A,I){2}
```

The value of the projection of the ring multiplication onto the the quotient space defined by an ideal I on a pair of the same classes $\langle x, x\rangle$ is denoted x^{2I}.

```
definition (in ring0) ideal_rsqr (_^2-) where
  x^2I ≡ ProjFun2(R, QuotientGroupRel(R,A,I), M)⟨x,x⟩
```

The class of the additive neutral element of the ring (i.e. 0) with respect to the equivalence relation defined by an ideal is the neutral of the projected addition.

```
lemma (in ring0) neutral_quotient:
  assumes I◁R
  shows
    QuotientGroupRel(R,A,I){0} = TheNeutralElement(QuotientBy(I),QuotientGroupOp(R,A,I))
  using ringAssum assms Group_ZF_2_4_L5B ideal_normal_add_subgroup QuotientBy_def
  unfolding IsAring_def by auto
```

Similarly, the class of the multiplicative neutral element of the ring (i.e. 1) with respect to the equivalence relation defined by an ideal is the neutral of the projected multiplication.

```
lemma (in ring0) one_quotient:
  assumes I◁R
  defines r ≡ QuotientGroupRel(R,A,I)
  shows r{1} = TheNeutralElement(QuotientBy(I),ProjFun2(R,r,M))
  using ringAssum assms Group_ZF_2_2_L1 ideal_equiv_rel quotientBy_mul_monoid
QuotientBy_def
```

```
  unfolding IsAring_def by auto
```

The class of 2 (i.e. $1 + 1$) is the same as the value of the addition projected on the quotient space on the pair of classes of 1.

```
lemma (in ring0) two_quotient:
  assumes I◁R
  defines r ≡ QuotientGroupRel(R,A,I)
  shows r{2} = QuotientGroupOp(R,A,I)⟨r{1},r{1}⟩
  using ringAssum assms EquivClass_1_L10 ideal_equiv_rel Ring_ZF_1_L2(2)
Ring_ZF_1_L2(2)
      Group_ZF_2_4_L5A ideal_normal_add_subgroup
  unfolding IsAring_def QuotientGroupOp_def by simp
```

The class of a square of an element of the ring is the same as the result of the projected multiplication on the pair of classes of the element.

```
lemma (in ring0) sqrt_quotient:
  assumes I◁R x∈R
  defines r ≡ QuotientGroupRel(R,A,I)
  shows r{x²} = ProjFun2(R,r, M)⟨r{x},r{x}⟩
  using assms EquivClass_1_L10 ideal_equiv_rel quotientBy_mul_monoid(1)
  by auto
```

The projection of the ring addition is distributive with respect to the projection of the ring multiplication.

```
lemma (in ring0) quotientBy_distributive:
  assumes I◁R
  defines r ≡ QuotientGroupRel(R,A,I)
  shows
    IsDistributive(QuotientBy(I),QuotientGroupOp(R,A,I),ProjFun2(R,r,M))
  using ringAssum assms QuotientBy_def ring_oper_distr(1) Ring_ZF_1_L4(1,3)

    quotientBy_mul_monoid(1) EquivClass_1_L10 ideal_equiv_rel Group_ZF_2_4_L5A

    ideal_normal_add_subgroup ring_oper_distr(2)
  unfolding quotient_def QuotientGroupOp_def IsAring_def IsDistributive_def

  by auto
```

The quotient group is a ring with the quotient multiplication.

```
theorem (in ring0) quotientBy_is_ring:
  assumes I◁R
  defines r ≡ QuotientGroupRel(R,A,I)
  shows IsAring(QuotientBy(I), QuotientGroupOp(R, A, I), ProjFun2(R,r,
M))
  using assms quotientBy_distributive quotientBy_mul_monoid(2) quotientBy_add_group
  unfolding IsAring_def by auto
```

An important property satisfied by many important rings is being Noetherian: every ideal is finitely generated.

```
definition (in ring0) isFinGen (_{is finitely generated}) where
  I◁R ⟹ I {is finitely generated} ≡ ∃S∈FinPow(R). I = ⟨S⟩_I
```

For Noetherian rings the arbitrary sum can be reduced to the sum of a finite subset of the initial set of ideals

```
theorem (in ring0) sum_ideals_noetherian:
  assumes ∀I∈𝓘. (I{is finitely generated}) 𝒥 ⊆ 𝓘
  shows ∃𝒯∈FinPow(𝒥). (⊕_I 𝒥) = (⊕_I 𝒯)
proof -
  from assms(2) have ⋃𝒥 ⊆ R using ideal_dest_subset by auto
  then have ⟨⋃𝒥⟩_I ◁R using generated_ideal_is_ideal by simp
  with assms(2) have (⊕_I 𝒥) ◁R using sumArbitraryIdeals_def
    by simp
  with assms(1) have (⊕_I 𝒥) {is finitely generated}
    using ideal_dest_subset by auto
  with ‹(⊕_I 𝒥)◁R›  obtain S where S∈FinPow(R) (⊕_I 𝒥) = ⟨S⟩_I
    using isFinGen_def by auto
  let N = λs∈S. {J∈FinPow(𝒥). s∈(⊕_I J)}
  from ‹S∈FinPow(R)› have Finite(S) unfolding FinPow_def
    by auto
  then have (∀t∈S. N(t) ≠ 0) ⟶
          (∃f. f ∈ (∏t∈S. N(t)) ∧ (∀t∈S. f(t) ∈ N(t)))
    using eqpoll_iff finite_choice AxiomCardinalChoiceGen_def
    unfolding Finite_def by blast
  moreover
  { fix t assume t∈S
    then have N(t) = {J∈FinPow(𝒥). t∈(⊕_I J)} using lamE by auto
    moreover
    from ‹S∈FinPow(R)› ‹(⊕_I 𝒥) = ⟨S⟩_I› ‹t∈S› have t∈(⊕_I 𝒥)
      using generated_ideal_contains_set unfolding FinPow_def by auto
    with assms(2) have ∃𝒯∈FinPow(𝒥). t∈(⊕_I 𝒯)
      using sum_ideals_finite_sum by simp
    ultimately have N(t)≠0 using assms(2) sum_ideals_finite_sum
      by auto
  }
  ultimately obtain f where
    f: f ∈ (∏t∈S. N(t)) ∀t∈S. f(t) ∈ N(t)
    by auto
  { fix y assume y ∈ f(S)
    with image_iff obtain x where x∈S and ⟨x,y⟩ ∈ f by auto
    with f(1) have y∈N(x) unfolding Pi_def by auto
    with ‹x∈S› have Finite(y) using lamE unfolding FinPow_def
      by simp
  }
  moreover
  from f(1) have f_Fun: f:S→ (⋃{N(t). t∈S})
    unfolding Pi_def Sigma_def by blast
  with ‹Finite(S)› have Finite(f(S))
    using Finite1_L6A Finite_Fin_iff Fin_into_Finite by blast
```

```
  ultimately have Finite(⋃(f(S))) using Finite_Union
    by auto
  with f_Fun f(2) have B: (⋃(f(S))) ∈ FinPow(𝒥)
    using func_imagedef lamE unfolding FinPow_def
    by auto
  then have (⋃(f(S))) ⊆ 𝒥 unfolding FinPow_def by auto
  with assms(2) have (⋃(f(S))) ⊆ ℐ  by auto
  hence sub: ⋃(⋃(f(S))) ⊆ R by auto
  { fix t assume t∈S
    with f(2) have f(t) ∈ FinPow(𝒥) t ∈ (⊕I(f(t)))
      using lamE by auto
    from assms(2) ‹f(t) ∈ FinPow(𝒥)› have f(t) ⊆ ℐ
      unfolding FinPow_def by auto
    from f_Fun ‹t∈S› have ⋃(ft) ⊆ ⋃(⋃(fS)) using func_imagedef
      by auto
    with sub have ⋃(f(t)) ⊆ ⟨⋃(⋃(f(S)))⟩I
      using generated_ideal_contains_set by blast
    with sub have ⟨⋃(ft)⟩I ⊆ ⟨⋃(⋃(fS))⟩I
      using generated_ideal_is_ideal generated_ideal_small by auto
    with ‹f(t) ⊆ ℐ› ‹t ∈ (⊕I(f(t)))› ‹(⋃(f(S))) ⊆ ℐ›
      have t ∈ (⊕I(⋃(f(S))))
      using sumArbitraryIdeals_def by auto
  } hence S ⊆ ⊕I(⋃(f(S))) by auto
  with sub ‹(⋃(f(S))) ⊆ ℐ› ‹(⊕I𝒥) = ⟨S⟩I› have (⊕I𝒥) ⊆ ⊕I(⋃(f(S)))
    using generated_ideal_small generated_ideal_is_ideal sumArbitraryIdeals_def

    by auto
  moreover
  from ‹⋃𝒥 ⊆ R› ‹(⋃(f(S))) ⊆ 𝒥› have ⋃(⋃(f(S))) ⊆ ⟨⋃𝒥⟩I
    using generated_ideal_contains_set by blast
  with assms(2) ‹(⋃(f(S))) ⊆ ℐ› ‹⟨⋃𝒥⟩I ◃R› have (⊕I(⋃(f(S)))) ⊆(⊕I𝒥)
    using generated_ideal_small sumArbitraryIdeals_def sumArbitraryIdeals_def

    by simp
  ultimately have (⊕I𝒥) = ⊕I(⋃(f(S))) by auto
  with B show thesis by auto
qed

end
```

49 Rings - Ideals of quotient rings

```
theory Ring_ZF_3 imports Ring_ZF_2 Group_ZF_5

begin
```

This section studies the ideals of quotient rings, and defines ring homomorphisms.

49.1 Ring homomorphisms

Morphisms in general are structure preserving functions between algebraic structures. In this section we study ring homomorphisms.

A ring homomorphism is a function between rings which has the morphism property with respect to both addition and multiplication operation, and maps one (the neutral element of multiplication) in the first ring to one in the second ring.

definition
```
ringHomomor(f,R,A,M,S,U,V) ≡ f:R→S ∧ IsMorphism(R,A,U,f) ∧ IsMorphism(R,M,V,f)

∧ f(TheNeutralElement(R,M)) = TheNeutralElement(S,V)
```

The next locale defines notation which we will use in this theory. We assume that we have two rings, one (which we will call the origin ring) defined by the triple (R, A, M) and the second one (which we will call the target ring) by the triple (S, U, V), and a homomorphism $f : R \to S$.

locale ring_homo =
  fixes R A M S U V f
  assumes origin: IsAring(R,A,M)
    and target: IsAring(S,U,V)
    and homomorphism: ringHomomor(f,R,A,M,S,U,V)

  fixes ringa (**infixl** $+_R$ 90)
  defines ringa_def [simp]: x$+_R$y ≡ A⟨x,y⟩

  fixes ringminus ($-_R$ _ 89)
  defines ringminus_def [simp]: ($-_R$x) ≡ GroupInv(R,A)(x)

  fixes ringsub (**infixl** $-_R$ 90)
  defines ringsub_def [simp]: x$-_R$y ≡ x$+_R$($-_R$y)

  fixes ringm (**infixl** $\cdot_R$ 95)
  defines ringm_def [simp]: x$\cdot_R$y ≡ M⟨x,y⟩

  fixes ringzero ($\mathbf{0}_R$)
  defines ringzero_def [simp]: $\mathbf{0}_R$ ≡ TheNeutralElement(R,A)

  fixes ringone ($\mathbf{1}_R$)
  defines ringone_def [simp]: $\mathbf{1}_R$ ≡ TheNeutralElement(R,M)

  fixes ringtwo ($\mathbf{2}_R$)
  defines ringtwo_def [simp]: $\mathbf{2}_R$ ≡ $\mathbf{1}_R$$+_R$$\mathbf{1}_R$

  fixes ringsq (_^{2R} [96] 97)
  defines ringsq_def [simp]: x^{2R} ≡ x$\cdot_R$x

```
  fixes ringas (infixl +S 90)
  defines ringas_def [simp]: x+Sb ≡ U⟨x,b⟩

  fixes ringminuss (-S _ 89)
  defines ringminuss_def [simp]: (-Sx) ≡ GroupInv(S,U)(x)

  fixes ringsubs (infixl -S 90)
  defines ringsubs_def [simp]: x-Sb ≡ x+S(-Sb)

  fixes ringms (infixl ·S 95)
  defines ringms_def [simp]: x·Sb ≡ V⟨ x,b⟩

  fixes ringzeros (0S)
  defines ringzeros_def [simp]: 0S ≡ TheNeutralElement(S,U)

  fixes ringones (1S)
  defines ringones_def [simp]: 1S ≡ TheNeutralElement(S,V)

  fixes ringtwos (2S)
  defines ringtwos_def [simp]: 2S ≡ 1S+S1S

  fixes ringsqs (_^2S [96] 97)
  defines ringsqs_def [simp]: x^2S ≡ x·Sx

  fixes rlistsum (∑ _ 70)
  defines rlistsum_def [simp]: ∑s ≡ Fold(A,0R,s)

  fixes rnat_mult (infix · 95)
  defines nat_mult_def [simp]: n·x ≡ ∑{⟨k,x⟩. k∈n}
```

We will write I◁R$_o$ to denote that I is an ideal of the ring R. Note that in this notation the R$_o$ part by itself has no meaning, only the whole ◁R$_o$ serves as postfix operator.

```
abbreviation (in ring_homo) ideal_origin (_◁Ro)
  where I◁Ro ≡ ring0.Ideal(R,A,M,I)
```

I◁R$_t$ means that I is an ideal of S.

```
abbreviation (in ring_homo) ideal_target (_◁Rt)
  where I◁Rt ≡ ring0.Ideal(S,U,V,I)
```

I◁$_p$R$_o$ means that I is a prime ideal of R.

```
abbreviation (in ring_homo) prime_ideal_origin (_◁pRo)
  where I◁pRo ≡ ring0.primeIdeal(R,A,M,I)
```

We will write I◁$_p$R$_t$ to denote that I is a prime ideal of the ring S.

```
abbreviation (in ring_homo) prime_ideal_target (_◁pRt)
  where I◁pRt ≡ ring0.primeIdeal(S,U,V,I)
```

ker denotes the kernel of f (which is assumed to be a homomorphism between R and S.

```
abbreviation (in ring_homo) kernel (ker 90) where
  ker ≡ f-{0_S}
```

The theorems proven in the ring0 context are valid in the ring_homo context when applied to the ring R.

```
sublocale ring_homo < origin_ring:ring0
  using origin unfolding ring0_def by auto
```

The theorems proven in the ring0 context are valid in the ring_homo context when applied to the ring S.

```
sublocale ring_homo < target_ring:ring0 S U V ringas ringminuss
  ringsubs ringms ringzeros ringones ringtwos ringsqs
  λs. Fold(U,0_S,s)
  λn x. Fold(U,0_S,{⟨k,x⟩. k∈n})
  using target unfolding ring0_def by auto
```

A ring homomorphism is a homomorphism both with respect to addition and multiplication.

```
lemma ringHomHom: assumes ringHomomor(f,R,A,M,S,U,V)
  shows Homomor(f,R,A,S,U) and Homomor(f,R,M,S,V)
  using assms unfolding ringHomomor_def Homomor_def
  by simp_all
```

Since in the ring_homo locale f is a ring homomorphism it implies that f is a function from R to S.

```
lemma (in ring_homo) f_is_fun: shows f:R→S
  using homomorphism unfolding ringHomomor_def by simp
```

In the ring_homo context A is the addition in the first (source) ring M is the multiplication there and U, V are the addition and multiplication resp. in the second (target) ring. The next lemma states the all these are binary operations, a trivial, but frequently used fact.

```
lemma (in ring_homo) AMUV_are_ops:
  shows A:R×R→R M:R×R→R U:S×S→S V:S×S→S
  using origin target
  unfolding IsAring_def IsAgroup_def IsAmonoid_def IsAssociative_def
  by auto
```

The kernel is a subset of R on which the value of f is zero (of the target ring)

```
lemma (in ring_homo) kernel_def_alt: shows ker = {r∈R. f(r) = 0_S}
  using f_is_fun func1_1_L14 by simp
```

the homomorphism f maps each element of the kernel to zero of the target ring.

```
lemma (in ring_homo) image_kernel:
  assumes x∈ker
  shows f(x) = 0_S
  using assms func1_1_L15 f_is_fun by simp
```

As a ring homomorphism f preserves multiplication.

```
lemma (in ring_homo) homomor_dest_mult:
  assumes x∈R y∈R
  shows f(x·_R y) = (f(x))·_S(f(y))
  using assms origin target homomorphism
  unfolding ringHomomor_def IsMorphism_def by simp
```

As a ring homomorphism f preserves addition.

```
lemma (in ring_homo) homomor_dest_add:
  assumes x∈R y∈R
  shows f(x+_R y) = (f(x))+_S(f(y))
  using homomor_eq origin target homomorphism assms
  unfolding IsAring_def ringHomomor_def IsMorphism_def
  by simp
```

For $x \in R$ the value of f is in S.

```
lemma (in ring_homo) homomor_val: assumes x∈R
  shows f(x) ∈ S
  using homomorphism assms apply_funtype unfolding ringHomomor_def
  by blast
```

A ring homomorphism preserves taking negative of an element.

```
lemma (in ring_homo) homomor_dest_minus:
  assumes x∈R
  shows f(-_R x) = -_S(f(x))
  using origin target homomorphism assms ringHomHom image_inv
  unfolding IsAring_def by auto
```

A ring homomorphism preserves substraction.

```
lemma (in ring_homo) homomor_dest_subs:
  assumes x∈R y∈R
  shows f(x-_R y) = (f(x))-_S(f(y))
  using assms homomor_dest_add homomor_dest_minus
  using origin_ring.Ring_ZF_1_L3(1) by auto
```

A ring homomorphism maps zero to zero.

```
lemma (in ring_homo) homomor_dest_zero:
  shows f(0_R) = 0_S
  using origin target homomorphism ringHomHom(1) image_neutral
  unfolding IsAring_def by auto
```

The kernel of a homomorphism is never empty.

```
lemma (in ring_homo) kernel_non_empty:
```

```
  shows 0_R ∈ker and ker≠0
  using homomor_dest_zero origin_ring.Ring_ZF_1_L2(1)
    func1_1_L15 f_is_fun by auto
```

The image of the kernel by f is the singleton $\{0_R\}$.

```
corollary (in ring_homo) image_kernel_2: shows f(ker) = {0_S}
proof -
  have f:R→S ker⊆R ker≠0 ∀x∈ker. f(x) = 0_S
    using f_is_fun kernel_def_alt kernel_non_empty by auto
  then show  f(ker) = {0_S} using image_constant_singleton
    by simp
qed
```

The inverse image of an ideal (in the target ring) is a normal subgroup of the addition group and an ideal in the origin ring. The kernel of the homomorphism is a subset of the inverse of image of every ideal.

```
lemma (in ring_homo) preimage_ideal:
  assumes J◁R_t
  shows
    IsAnormalSubgroup(R,A,f-(J))
    (f-(J))◁R_o ker ⊆ f-(J)
proof -
  from origin target homomorphism assms
    show IsAnormalSubgroup(R,A,f-(J))
      using ringHomHom(1) preimage_normal_subgroup
        target_ring.ideal_normal_add_subgroup
      unfolding IsAring_def by blast
  then have IsAsubgroup(f-(J),A) unfolding IsAnormalSubgroup_def
    by auto
  moreover
  { fix x y assume x ∈ f-(J) y∈R
    from homomorphism have f:R→S
      unfolding ringHomomor_def by simp
    with ‹x ∈ f-(J)› have x∈R and f(x) ∈ J
      using func1_1_L15 unfolding ringHomomor_def
      by simp_all
    with assms ‹y∈R› ‹f:R→S› have
      V⟨f(x),f(y)⟩ ∈ J and V⟨f(y),f(x)⟩∈J
      using apply_funtype target_ring.ideal_dest_mult by simp_all
    with ‹x∈R› ‹y∈R› have f(M⟨x,y⟩) ∈ J and f(M⟨y,x⟩) ∈ J
      using homomor_dest_mult by auto
    with ‹x∈R› ‹y∈R› ‹f:R→S› have
      M⟨x,y⟩ ∈ f-(J) and M⟨y,x⟩ ∈ f-(J)
      using origin_ring.Ring_ZF_1_L4(3) func1_1_L15 by auto
  }
  ultimately show (f-(J))◁R_o using origin_ring.Ideal_def
    by auto
  from assms show ker ⊆ f-(J) using target_ring.ideal_dest_zero
    by auto
```

```
qed
```

Kernel of the homomorphism in an ideal.

```
lemma (in ring_homo) kernel_ideal: shows ker ◁R_o
  using target_ring.zero_ideal preimage_ideal(2) by simp
```

The inverse image of a prime ideal by a homomorphism is not the whole ring. Proof by contradiction.

```
lemma (in ring_homo) vimage_prime_ideal_not_all:
  assumes J◁_pR_t shows f-(J) ≠ R
proof -
  { assume R = f-(J)
    then have R = {x∈R. f(x)∈J} using f_is_fun func1_1_L15
      by simp
    then have 1_R ∈ {x∈R. f(x)∈J} using origin_ring.Ring_ZF_1_L2(2)
      by simp
    with origin_ring.ringAssum target_ring.ringAssum homomorphism assms
      have False using target_ring.ideal_with_one
        unfolding target_ring.primeIdeal_def ringHomomor_def by auto
  } thus f-(J) ≠ R by blast
qed
```

Even more, if the target ring of the homomorphism is commutative and the ideal is prime then its preimage is also. Note that this is not true in general.

```
lemma (in ring_homo) preimage_prime_ideal_comm:
  assumes J◁_pR_t V {is commutative on} S
  shows (f-(J))◁_pR_o
proof -
  have ∀x∈R. ∀y∈R. (∀z∈R. x·_Rz·_Ry ∈ f-(J)) ⟶ x∈f-(J) ∨ y∈f-( J)
  proof -
    { fix x y assume x∈R y∈R and as: ∀z∈R. x·_Rz·_Ry ∈ f-(J)
        and y ∉ f-(J)
      { fix s assume s∈S
        with assms(2) ‹x∈R› ‹y∈R› have
          (f(x))·_Ss·_S(f(y)) = s·_S((f(x))·_S(f(y)))
          using f_is_fun apply_funtype target_ring.Ring_ZF_1_L11(2)
          unfolding IsCommutative_def by auto
        with ‹x∈R› ‹y∈R› have (fx)·_Ss·_S(fy) = s·_S(f(x·_Ry))
          using homomor_dest_mult by simp
        with assms(1) ‹s∈S› ‹x∈R› as have (f(x))·_Ss·_S(f(y)) ∈ J
          using origin_ring.Ring_ZF_1_L2(2) origin_ring.Ring_ZF_1_L3(5)
            func1_1_L15 f_is_fun target_ring.ideal_dest_mult(2)
          unfolding target_ring.primeIdeal_def by auto
      } hence ∀z∈S. (fx) ·_S z ·_S (fy) ∈ J by simp
      with assms(1) ‹x∈R› ‹y∈R›  have f(x)∈ J ∨ f(y)∈J
        using target_ring.equivalent_prime_ideal f_is_fun apply_funtype

          by simp
```

```
      with ‹x∈R› ‹y∈R› ‹y ∉ f-(J)› have x∈f-(J)
        using func1_1_L15 f_is_fun by simp
    } thus thesis by blast
  qed
  moreover from assms have (f-J)◁Rₒ using preimage_ideal
    unfolding target_ring.primeIdeal_def by auto
  moreover from assms(1) have f-(J) ≠ R using vimage_prime_ideal_not_all

    by simp
  ultimately show (f-(J))◁ₚRₒ
    by (rule origin_ring.equivalent_prime_ideal_2)
qed
```

We can replace the assumption that the target ring of the homomorphism is commutative with the assumption that homomorphism is surjective in `preimage_prime_ideal_comm` above and we can show the same assertion that the preimage of a prime ideal prime.

```
lemma (in ring_homo) preimage_prime_ideal_surj:
  assumes J◁ₚRₜ f ∈ surj(R,S)
  shows (f-(J))◁ₚRₒ
proof -
  have ∀x∈R. ∀y∈R. (∀z∈R. x·ᵣz·ᵣy ∈ f-(J)) ⟶ x∈f-(J) ∨ y∈f-(J)
  proof -
    { fix x y assume x∈R y∈R and as: ∀z∈R. x·ᵣz·ᵣy ∈ f-(J)
        and y ∉ f-(J)
      { fix s assume s: s∈S
        with assms(2) obtain t where s=f(t) t∈R
          unfolding surj_def by auto
        with ‹x∈R› ‹y∈R› as have (fx)·ₛs·ₛ(fy)∈J
          using homomor_dest_mult origin_ring.Ring_ZF_1_L4(3)
            func1_1_L15 f_is_fun by simp
      } hence ∀z∈S. (f(x)) ·ₛ z ·ₛ (f(y)) ∈ J by simp
      with target_ring.equivalent_prime_ideal assms(1) ‹x∈R› ‹y∈R›
      have f(x)∈J∨f(y)∈J using f_is_fun apply_funtype
        by auto
      with ‹x∈R› ‹y∈R› ‹y ∉ f-(J)› have x∈f-(J)
        using func1_1_L15 f_is_fun by auto
    } thus thesis by blast
  qed
  moreover from assms have (f-(J))◁Rₒ using preimage_ideal
    unfolding target_ring.primeIdeal_def by auto
  moreover from assms(1) have f-(J) ≠ R using vimage_prime_ideal_not_all

    by simp
  ultimately show (f-(J))◁ₚRₒ
    by (rule origin_ring.equivalent_prime_ideal_2)
qed
```

49.2 Quotient ring with quotient map

The notion of a quotient ring (a.k.a factor ring, difference ring or residue class) is analogous to the notion of quotient group from the group theory.

The next locale `ring2` extends the `ring0` locale (defined in the `Ring_ZF` theory) with the assumption that some fixed set I is an ideal. It also defines some notation related to quotient rings, in particular we define the function (projection) f_I that maps each element r of the ring R to its class $r_I(\{r\}$ where r_I is the quotient group relation defined by I as a (normal) subgroup of R with addition.

```
locale ring2 = ring0 +
  fixes I
  assumes idealAssum: I◁R

  fixes quot (R_I)
  defines quot_def [simp]: R_I ≡ QuotientBy(I)

  fixes qrel (r_I)
  defines qrel_def [simp]: r_I ≡ QuotientGroupRel(R,A,I)

  fixes qfun (f_I)
  defines qfun_def [simp]: f_I ≡ λr∈R. r_I{r}

  fixes qadd (A_I)
  defines qadd_def [simp]: A_I ≡ QuotientGroupOp(R, A, I)

  fixes qmul (M_I)
  defines qmul_def [simp]: M_I ≡ ProjFun2(R, qrel, M)
```

The expression `J◁R_I` will mean that J is an ideal of the quotient ring R_I (with the quotient addition and multiplication).

```
abbreviation (in ring2) qideal (_◁R_I) where
  J◁R_I ≡ ring0.Ideal(R_I,A_I,M_I,J)
```

In the `ring2` The expression `J◁_pR_I` means that J is a prime ideal of the quotient ring R_I.

```
abbreviation (in ring2) qprimeIdeal (_◁_pR_I) where
  J◁_pR_I ≡ ring0.primeIdeal(R_I,A_I,M_I,J)
```

Theorems proven in the `ring0` context can be applied to the quotient ring in the `ring2` context.

```
sublocale ring2 < quotient_ring: ring0 quot qadd qmul
  λx y. ideal_radd(x,I,y) λy. ideal_rmin(I,y)
  λx y. ideal_rsub(x,I,y) λx y. ideal_rmult(x,I,y)
  0_I 1_I 2_I λx. (x^{2I})
  λs. Fold(A_I, 0_I, s)
```

```
  λn x.  Fold(A_I, 0_I, {⟨k, x⟩ . k ∈ n})
  using quotientBy_is_ring idealAssum neutral_quotient
    one_quotient two_quotient
  unfolding ring0_def  ideal_radd_def ideal_rmin_def
    ideal_rsub_def ideal_rmult_def ideal_rzero_def
    ideal_rone_def ideal_rtwo_def ideal_rsqr_def by auto
```

The quotient map is a homomorphism of rings. This is probably one of the most sophisticated facts in IsarMathlib that Isabelle's simp method proves from 10 facts and 5 definitions.

```
theorem (in ring2) quotient_fun_homomor:
  shows ringHomomor(f_I,R,A,M,R_I,A_I,M_I)
  using ringAssum idealAssum ideal_normal_add_subgroup add_group.quotient_map

      Ring_ZF_1_L4(3) EquivClass_1_L10 Ring_ZF_1_L2(2) Group_ZF_2_2_L1

      ideal_equiv_rel quotientBy_mul_monoid(1) QuotientBy_def
  unfolding IsAring_def Homomor_def IsMorphism_def ringHomomor_def
  by simp
```

The quotient map is surjective

```
lemma (in ring2) quot_fun:
  shows f_I∈surj(R,R_I)
  using lam_funtype idealAssum QuotientBy_def
  unfolding quotient_def surj_def by auto
```

The theorems proven in the `ring_homo` context are valid in the `ring_homo` context when applied to the quotient ring as the second (target) ring and the quotient map as the ring homomorphism.

```
sublocale ring2 < quot_homomorphism: ring_homo R A M quot qadd qmul qfun
  _ _ _ _ _ _ _ _ _ λx y. ideal_radd(x,I,y) λy. ideal_rmin(I,y)
  λx y. ideal_rsub(x,I,y) λx y. ideal_rmult(x,I,y)
  0_I 1_I 2_I λx. (x^2I)
  using ringAssum quotient_ring.ringAssum quotient_fun_homomor
  unfolding ring_homo_def by simp_all
```

The ideal we divide by is the kernel of the quotient map.

```
lemma (in ring2) quotient_kernel:
  shows quot_homomorphism.kernel = I
proof
  from idealAssum show quot_homomorphism.kernel ⊆ I
    using add_group.Group_ZF_2_4_L5E ideal_normal_add_subgroup
        neutral_quotient QuotientBy_def
        quot_homomorphism.image_kernel func1_1_L15
    unfolding  ideal_rzero_def QuotientBy_def by auto
  { fix t assume t∈I
    then have t∈R using ideal_dest_subset idealAssum by auto
    with idealAssum ‹t∈I› have t ∈ f_I-{0_I}
```

```
      using add_group.Group_ZF_2_4_L5E ideal_normal_add_subgroup
        neutral_quotient QuotientBy_def func1_1_L15 surj_is_fun
      unfolding ideal_rzero_def by auto
  } thus I ⊆ quot_homomorphism.kernel by blast
qed
```

If an ideal I is a subset of the kernel of the homomorphism then the image of the ideal generated by $I \cup J$, where J is another ideal, is the same as the image of J. Note that `J+_I I` notation means the ideal generated by the union of ideals J and J, see the definitions of `sumIdeal` and `generatedIdeal` in the `Ring_ZF_2` theory, and also corollary `sum_ideals_is_sum_elements` for an alternative definition.

```
theorem (in ring_homo) kernel_empty_image:
  assumes J◁R I ⊆ ker I◁R
  shows f(J+_I I) = f(J) f(I+_I J) = f(J)
proof-
  from assms(1,3) have J+_I I ⊆ R
    using origin_ring.sum_ideals_is_ideal
      origin_ring.ideal_dest_subset by auto
  show f(J+_I I) = f(J)
  proof
    { fix t assume t∈f(J+_I I)
      with ‹J+_I I ⊆ R› obtain q where q: q ∈ J+_I I t=f(q)
        using func_imagedef f_is_fun by auto
      with assms(1,3) ‹q ∈ J+_I I› have q∈A(J×I) J×I ⊆ R×R
        using origin_ring.sum_ideals_is_sum_elements
        assms(1,3) origin_ring.ideal_dest_subset by auto
      from origin_ring.add_group.groupAssum ‹J×I ⊆ R×R›
      have A(J×I) = {A(p). p∈J×I}
        unfolding IsAgroup_def IsAmonoid_def IsAssociative_def
        using func_imagedef by auto
      with ‹q∈A(J×I)› obtain q_j q_i where q_j∈J q_i∈I q=q_j+_R q_i
        by auto
      with assms have f(q) = (f(q_j)) +_S 0_S
        using homomor_dest_add origin_ring.ideal_dest_subset image_kernel
        by blast
      moreover from assms(1) ‹q_j∈J› have f(q_j) ∈ S
        using origin_ring.ideal_dest_subset f_is_fun apply_funtype by
blast
      ultimately have f(q) = f(q_j) using target_ring.Ring_ZF_1_L3(3)
        by auto
      with assms ‹t=f(q)› ‹q_j∈J› have t ∈ f(J)
        using func_imagedef f_is_fun origin_ring.ideal_dest_subset by
auto
    } thus f(J+_I I) ⊆ f(J) by blast
    { fix t assume t∈f(J)
      with assms(1) obtain q where q:q∈J t=f(q)
        using func_imagedef f_is_fun origin_ring.ideal_dest_subset
        by auto
```

```
      from ‹q∈J› have q∈J∪I by auto
      moreover from assms(1,3) have J∪I ⊆ R
        using origin_ring.ideal_dest_subset by auto
      ultimately have q∈⟨J∪I⟩_I using origin_ring.generated_ideal_contains_set

        by auto
      with assms(1,3) ‹J+_I I ⊆ R› ‹t=f(q)› have t∈f(J+_I I)
        using origin_ring.sumIdeal_def f_is_fun func_imagedef by auto
    } thus f(J) ⊆ f(J+_I I) by auto
  qed
  with assms(1,3) show f (I+_I J) = f(J)
    using origin_ring.sum_ideals_commute by auto
qed
```

49.3 Quotient ideals

If we have an ideal J in a ring R, and another ideal I contained in J, then we can form the quotient ideal J/I whose elements are of the form $a + I$ where a is an element of J.

The preimage of an ideal is an ideal, so it applies to the quotient map; but the preimage ideal contains the quotient ideal.

```
lemma (in ring2) ideal_quot_preimage:
  assumes J◁R_I
  shows (f_I-(J)) ◁R I ⊆ f_I-(J)
proof -
  from assms quot_homomorphism.preimage_ideal show (f_I-(J)) ◁R
    by simp
  { fix x assume x: x∈I
    with idealAssum have x∈R using ideal_dest_subset by auto
    from assms ‹x∈I› have f_I(x) ∈ J
      using quotient_kernel quot_homomorphism.image_kernel
        quotient_ring.ideal_dest_zero by simp
    with ‹x∈R› have x∈f_I-(J) using quot_homomorphism.f_is_fun func1_1_L15

      by simp
  } thus I ⊆ f_I-(J) by auto
qed
```

Since the map is surjective, the image is also an ideal

```
lemma (in ring_homo) image_ideal_surj:
  assumes J◁R_o f∈surj(R,S)
  shows (f(J)) ◁R_t
proof -
  from assms homomorphism target_ring.ringAssum origin_ring.ringAssum

  have IsAsubgroup(f(J),U)
    using ringHomHom(1) image_subgroup f_is_fun
```

```
    unfolding IsAring_def origin_ring.Ideal_def by blast
  moreover
  { fix x y assume x∈f(J) y∈S
    from assms(1) ‹x∈f(J)› obtain j where x=f(j) j∈J
      using func_imagedef f_is_fun origin_ring.ideal_dest_subset
      by auto
    from assms ‹y∈S› ‹j∈J› have j∈R and y∈f(R)
      using origin_ring.ideal_dest_subset surj_range_image_domain
      by auto
    with assms(1) obtain s where y=f(s) s∈R
      using func_imagedef origin_ring.ideal_dest_subset f_is_fun
      by auto
    with assms(1) ‹j∈R› ‹x=f(j)› ‹x∈f(J)› have
      V⟨x,y⟩ = f(M⟨j,s⟩) and V⟨y,x⟩ = f(M⟨s,j⟩)
      using homomor_dest_mult origin_ring.ideal_dest_subset by auto
    with assms(1) ‹j∈J› ‹s∈R› have (x·_S y)∈f(J) and (y·_S x)∈fJ
      using origin_ring.ideal_dest_mult func_imagedef f_is_fun
        origin_ring.ideal_dest_subset by auto
  } hence ∀x∈f(J). ∀y∈S. (y·_S x) ∈ f(J) ∧ (x·_S y) ∈ f(J)
    by auto
  ultimately show thesis unfolding target_ring.Ideal_def by simp
qed
```

If the homomorphism is a surjection and given two ideals in the target ring the inverse image of their product ideal is the sum ideal of the product ideal of their inverse images and the kernel of the homomorphism.

```
corollary (in ring_homo) prime_ideal_quot:
  assumes J◁R_t K◁R_t f∈surj(R,S)
  shows f-(target_ring.productIdeal(J, K)) =
  origin_ring.sumIdeal(origin_ring.productIdeal((f-(J)),(f-(K))), ker)
proof
  let P = origin_ring.sumIdeal(origin_ring.productIdeal((f-(J)),(f-(K))),
f-{0_S})
  let Q = target_ring.productIdeal(J, K)
  from assms(1,2) have ideals: (f-(J)) ◁R_o (f-(K)) ◁R_o
    using preimage_ideal by auto
  then have idealJK: ((f-(J))·_I(f-(K))) ◁R_o
    using origin_ring.product_in_intersection(2) by auto
  then have P ◁R_o and P⊆R
    using kernel_ideal origin_ring.sum_ideals_is_ideal
      origin_ring.ideal_dest_subset by simp_all
  with assms(3) have  (f(P)) ◁R_t using  image_ideal_surj
    by simp
  let X = ((f-(J)) ·_I (f-(K)))∪(f-{0_S})
  from assms(3) idealJK have X ⊆ R
      using func1_1_L6A surj_is_fun origin_ring.ideal_dest_subset
      by blast
  with idealJK have X ⊆ P
      using kernel_ideal origin_ring.sumIdeal_def
```

```
      origin_ring.generated_ideal_contains_set by simp
  { fix t assume t∈V (J×K)
    moreover from assms have J×K ⊆ S×S
      using target_ring.ideal_dest_subset by blast
    ultimately obtain j k where j∈J k∈K t=V⟨j,k⟩
      using func_imagedef AMUV_are_ops(4) by auto
    from assms(1) ‹j∈J› have j∈S
      using target_ring.ideal_dest_subset by blast
    with assms(3) obtain j₀ where  j₀∈R f(j₀) = j
      unfolding surj_def by auto
    with assms(2,3) ‹k∈K› obtain k₀ where k₀∈R f(k₀) = k
      using target_ring.ideal_dest_subset unfolding surj_def
      by blast
    with ‹t=V⟨j,k⟩› ‹f(j₀) = j›  ‹j₀∈R› have t=f(M⟨j₀,k₀⟩)
      using homomor_dest_mult by simp
    from assms(3) have (f-(J))×(f-(K)) ⊆ R×R
      using func1_1_L6A  f_is_fun by blast
    moreover from assms(3) ‹j₀∈R› ‹f(j₀) = j› ‹j∈J› ‹k∈K› ‹k₀∈R› ‹f(k₀)
= k›
      have ⟨j₀,k₀⟩ ∈ (f-J)×(f-K)
      using func1_1_L15 unfolding surj_def by auto
    ultimately have M⟨j₀,k₀⟩ ∈ M((f-(J))×(f-(K)))
      using AMUV_are_ops func_imagedef by auto
    with ideals ‹X ⊆ P› have M⟨j₀,k₀⟩ ∈ P
      using origin_ring.product_in_intersection(3)
      by blast
    with ‹P⊆R› ‹t=f(M⟨j₀,k₀⟩)› have t∈(f(P))
      using f_is_fun func1_1_L15D by simp
  } hence V(J × K) ⊆ f(P) by blast
  with assms(1,2) ‹(f(P)) ◁Rₜ› have Q⊆f(P)
    using target_ring.generated_ideal_small target_ring.productIdeal_def
    by simp
  then have R: f-(Q) ⊆ f-(fP)
    unfolding vimage_def by blast
  from ‹X ⊆ P› ‹P ◁Rₒ› ‹P⊆R› have P_ideal: P ∈ {N∈ℐ. f-{0_S} ⊆ N}
    by auto
  { fix t assume t∈f-(f(P))
    then have f(t) ∈ f(P) and t∈R
      using func1_1_L15 f_is_fun by simp_all
    with P_ideal obtain s where f(t) = f(s) s∈P
      using func_imagedef f_is_fun by auto
    from P_ideal ‹s∈P› have s∈R by blast
    from ‹t∈R› ‹s∈R› ‹f(t) = f(s)› have f(t-_Rs) = 0_S
      using f_is_fun apply_funtype target_ring.Ring_ZF_1_L3(7)
        homomor_dest_subs by simp
    with ‹t∈R› ‹s∈R› ‹X⊆P› have t-_Rs ∈ P
      using origin_ring.Ring_ZF_1_L4(2) func1_1_L15 f_is_fun
      by auto
    with P_ideal ‹s∈P› have s+_R(t-_Rs) ∈ P
```

```
      using origin_ring.ideal_dest_sum by auto
    with ‹t∈R› ‹s∈R› have t∈P
      using origin_ring.Ring_ZF_2_L1A(5) by auto
  } with R show f-(Q) ⊆ P
    by auto
  { fix t assume as: t ∈ M((f-(J))×(f-(K)))
    have (f-(J))×(f-(K)) ⊆ R×R
      using func1_1_L15 f_is_fun by auto
    with as obtain t_j t_k where
      jk: t=t_j·_R t_k t_j∈f-(J) t_k∈f-(K)
      using AMUV_are_ops(2) func_imagedef IsAssociative_def
      by auto
    from as have t∈R using AMUV_are_ops(2) func1_1_L6(2) by blast
    from jk(2,3) have t_j∈R f(t_j)∈J and t_k∈R f(t_k)∈K
      using func1_1_L15 f_is_fun by simp_all
    from jk(1) ‹t_j∈R› ‹t_k∈R› have ft: f(t) = ((f(t_j))·_S(f(t_k)))
      using homomor_dest_mult by simp
    from ‹f(t_j)∈J› ‹f(t_k) ∈K› have ⟨f(t_j),f(t_k)⟩ ∈ J×K
      by simp
    moreover from assms have V:S×S→S and J×K ⊆ S×S
      using AMUV_are_ops(4) target_ring.ideal_dest_subset
      by auto
    ultimately have V⟨f(t_j), f(t_k)⟩ ∈V(J×K)
      using func_imagedef by force
    with assms ft ‹t∈R› have t ∈ f-(Q)
      using target_ring.product_in_intersection(3) func1_1_L15 f_is_fun

      by auto
  } hence M(f-(J)×f-(K))⊆f-(Q)
    by auto
  moreover from assms(1,2) have
    id: (f-(Q)) ◁R_o
    using preimage_ideal target_ring.product_in_intersection(2)
    by simp
  ultimately have (f - J) ·_I (f - K)⊆f-(Q)
    using ideals origin_ring.generated_ideal_small origin_ring.productIdeal_def
    by auto
  with assms(1,2) have X ⊆ f-(Q)
    using target_ring.product_in_intersection(2) target_ring.ideal_dest_zero

    by auto
  with idealJK id show
    ((f-(J))·_I(f-(K)))+_I(f-{0_S}) ⊆ f-(Q)
    using origin_ring.generated_ideal_small kernel_ideal origin_ring.sumIdeal_def
    by simp
qed
```

If the homomorphism is surjective then the product ideal of ideals J, K in the target ring is the image of the product ideal (in the source ring) of the inverse images of J, K.

```
corollary (in ring_homo) prime_ideal_quot_2:
  assumes J⊲Rt K⊲Rt f∈surj(R,S)
  shows target_ring.productIdeal(J, K) =
    f(origin_ring.productIdeal((f-(J)), (f-(K))))
proof -
  from assms have f(f-(target_ring.productIdeal(J, K)))
      = f(((f-(J)) ·I (f-(K))) +I (ker))
    using prime_ideal_quot by simp
  with assms show thesis
    using target_ring.product_in_intersection(2)
      target_ring.ideal_dest_subset surj_image_vimage
      origin_ring.product_in_intersection(2) preimage_ideal
      kernel_empty_image(1) target_ring.zero_ideal
    by simp
qed
```

If the homomorphism is surjective and an ideal in the source ring contains the kernel, then the image of that ideal is a prime ideal in the target ring.

```
lemma (in ring_homo) preimage_ideal_prime:
  assumes J⊲pRo ker ⊆ J f∈surj(R,S)
  shows (f(J))⊲pRt
proof -
  from assms(1) have J ⊆ R J ≠ R
    unfolding origin_ring.primeIdeal_def
    using origin_ring.ideal_dest_subset by auto
  from assms(1,3) have (f(J))⊲Rt
    using image_ideal_surj  unfolding origin_ring.primeIdeal_def
    by auto
  moreover
  { assume f(J) = S
    from ‹J ⊆ R› ‹J ≠ R› obtain t where t∈R-J by auto
    with assms(3) have f(t)∈S
      using apply_funtype f_is_fun by auto
    with assms(3) ‹J ⊆ R› ‹f(J) = S› obtain j
      where j: j∈J f(t)= f(j)
      using func_imagedef f_is_fun by auto
    from ‹j∈J› ‹J ⊆ R› have j∈R by auto
    with assms(3) ‹f(t)= f(j)› ‹t∈R-J› have t-Rj∈f-{0S}
      using apply_type target_ring.Ring_ZF_1_L3(7) f_is_fun
        homomor_dest_subs origin_ring.Ring_ZF_1_L4(2) func1_1_L15
        by auto
    with assms(1,2) j(1) have j+R(t-Rj) ∈J
      unfolding origin_ring.primeIdeal_def
      using origin_ring.ideal_dest_sum by auto
    with ‹j∈R› ‹t∈R-J› have False using origin_ring.Ring_ZF_2_L1A(5)
      by auto
  } hence f(J) ≠ S by auto
  moreover
```

```
  { fix I K assume as: I∈target_ring.ideals K∈target_ring.ideals
          target_ring.productIdeal(I, K) ⊆ f(J)
    let A = (f-(I)) ·I (f-(K))
    from as(1,2) have A ⊆ f-(f(A))
      using origin_ring.product_in_intersection(2) preimage_ideal(2)
        origin_ring.ideal_dest_subset func1_1_L9 f_is_fun by auto
    with assms(3) as have A ⊆ f-(f(J))
      using prime_ideal_quot_2 vimage_mono by force
    moreover from assms(1) have J ⊆ f-(f(J))
      using func1_1_L9 f_is_fun origin_ring.ideal_dest_subset
      unfolding origin_ring.primeIdeal_def by auto
    moreover
    { fix t assume t ∈ f-(f(J))
      then have f(t)∈f(J) t∈R using func1_1_L15 f_is_fun
        by auto
      from assms(1) ‹f(t)∈f(J)› obtain s where f(t)=f(s) s∈J
        using func_imagedef f_is_fun origin_ring.ideal_dest_subset
        unfolding origin_ring.primeIdeal_def by auto
      from assms(1) ‹s∈J› have s∈R
        unfolding origin_ring.primeIdeal_def
        using origin_ring.ideal_dest_subset by auto
      with assms(2) ‹f(t)=f(s)› ‹t∈R› have t-Rs ∈ J
        using target_ring.Ring_ZF_1_L3(7) apply_funtype f_is_fun
          homomor_dest_subs origin_ring.Ring_ZF_1_L4(2) func1_1_L15
        by auto
      with assms(1) ‹s∈J› have s+R(t-Rs)∈J
        using origin_ring.ideal_dest_sum
        unfolding origin_ring.primeIdeal_def by auto
      with ‹s∈R› ‹t∈R› have t∈J using origin_ring.Ring_ZF_2_L1A(5)
        by auto
    } hence f-(fJ) ⊆ J by auto
    ultimately have A ⊆ J by auto
    with assms(1) as(1,2) have (f-(I)) ⊆ J ∨ (f-(K)) ⊆ J
      using preimage_ideal origin_ring.ideal_dest_subset
      unfolding origin_ring.primeIdeal_def
      by auto
    hence f(f-(I)) ⊆ fJ ∨ f(f-(K)) ⊆ f(J)
      by auto
    with assms(3) as(1,2) have I ⊆ f(J) ∨ K ⊆ f(J)
      using surj_image_vimage by auto
  }
  ultimately show thesis unfolding target_ring.primeIdeal_def by auto
qed
```

The ideals of the quotient ring are in bijection with the ideals of the original ring that contain the ideal by which we made the quotient.

```
theorem (in ring_homo) ideal_quot_bijection:
  assumes f∈surj(R,S)
  defines idealFun ≡ λJ∈target_ring.ideals. f-(J)
```

```
  shows idealFun ∈ bij(target_ring.ideals,{K∈ℐ. ker ⊆ K})
proof -
  let ℐt = target_ring.ideals
  have idealFun : ℐt → {f-(J). J∈ℐt}
    unfolding idealFun_def using lam_funtype by simp
  moreover
  { fix t assume t ∈ {f-(J). J∈ℐt}
    then obtain K where K ∈ ℐt f-(K) = t by auto
    then have K◁Rt by simp
    then have (f-(K))◁R ker ⊆ f-(K) using preimage_ideal(2,3)
      by simp_all
    with ‹f-(K) = t› have ker ⊆ t t◁R by simp_all
    with ‹f-(K) = t› have t∈{K∈ℐ. ker ⊆ K} using func1_1_L3 f_is_fun

      by blast
  } hence {f-(J). J∈ℐt} ⊆ {K∈ℐ. ker ⊆ K} by blast
  ultimately have I: idealFun : ℐt → {K∈ℐ. ker ⊆ K}
    using func1_1_L1B by auto
  { fix w x assume
      as: w◁Rt x◁Rt w⊆S x⊆S idealFun(w) = idealFun(x)
    then have f(f-(w)) = f(f-(x)) unfolding idealFun_def by simp
    with assms(1) as have w = x using surj_image_vimage
      by simp
  }
  with I have idealFun ∈  inj(ℐt,{K∈ℐ. ker ⊆ K})
    unfolding inj_def by auto
  moreover
  { fix y assume y◁R y⊆R ker ⊆ y
    from ‹y⊆R› have y ⊆ f-(fy) using func1_1_L9 f_is_fun
      by auto
    moreover
    { fix t assume t∈f-(f(y))
      then have f(t) ∈ f(y) t∈R using func1_1_L15 f_is_fun
        by auto
      from ‹f(t) ∈ f(y)› ‹y⊆R› obtain s where s∈y f(t) = f(s)
        using func_imagedef f_is_fun by auto
      from ‹s∈y›  ‹y⊆R› have s∈R by auto
      with ‹t∈R› ‹f(t) = f(s)› ‹ker ⊆ y› have t-Rs ∈ y
        using target_ring.Ring_ZF_1_L3(7) homomor_dest_subs
          origin_ring.Ring_ZF_1_L4(2) func1_1_L15 f_is_fun
        by auto
      with ‹s∈y› ‹y◁R› ‹s∈R› ‹t∈R› have t∈y
        using origin_ring.ideal_dest_sum origin_ring.Ring_ZF_2_L1A(5)
        by force
    }
    ultimately have f-(f(y)) = y by blast
    moreover have f(y) ⊆ S using func1_1_L6(2) f_is_fun
      unfolding surj_def by auto
    moreover from assms(1) ‹y◁R› have (f(y))◁Rt using image_ideal_surj
```

```
      by auto
    ultimately have ∃x∈ℐₜ. idealFun(x) = y
      unfolding idealFun_def by auto
  }
  with I have idealFun ∈ surj(ℐₜ,{K∈ℐ. ker ⊆ K})
    unfolding surj_def by auto
  ultimately show thesis unfolding bij_def by blast
qed
```

Assume the homomorphism f is surjective and consider the function that maps an ideal J in the target ring to its inverse image $f^{-1}(J)$ (in the source ring). Then the value of the converse of that function on any ideal containing the kernel of f is the image of that ideal under the homomorphism f.

```
theorem (in ring_homo) quot_converse:
  defines F ≡ λJ∈target_ring.ideals. f-(J)
  assumes J◁R ker⊆J f∈surj(R,S)
  shows converse(F)(J) = f(J)
proof-
  let g = λJ∈{K∈ℐ. ker⊆K}. f(J)
  let ℐₜ = target_ring.ideals
  let C_F = converse(F)
  from assms(1,4) have C_F ∈ bij({K∈ℐ. ker⊆K}, ℐₜ)
    using bij_converse_bij ideal_quot_bijection
    by auto
  then have I: C_F:{K∈ℐ. ker⊆K} → ℐₜ
    unfolding bij_def inj_def by auto
  moreover from assms(2,3) have J: J ∈ {K∈ℐ. ker⊆K}
    using origin_ring.ideal_dest_subset by auto
  ultimately have ideal: C_F(J) ∈ ℐₜ
    using apply_funtype by blast
  with assms(1,4) ideal have g(F(C_F(J))) = C_F(J)
    using preimage_ideal origin_ring.ideal_dest_subset
      surj_image_vimage by auto
  moreover
  from assms(1) have F: ℐₜ → {f-(J). J∈ℐₜ}
    using lam_funtype by simp
  with I J have F(C_F(J)) = J
    using right_inverse_lemma by simp
  ultimately have g(J) = C_F(J) by simp
  with J show thesis by simp
qed
```

Since the map is surjective, this bijection restricts to prime ideals on both sides.

```
corollary (in ring_homo) prime_ideal_quot_3:
  assumes K◁Rₜ f ∈ surj(R,S)
  shows K◁ₚRₜ ⟷ ((f-(K))◁ₚR)
proof
  { assume K◁ₚRₜ
```

```
    with assms(2) show (f-(K))◁pR
      using preimage_prime_ideal_surj target_ring.ideal_dest_subset
      unfolding target_ring.primeIdeal_def by auto
  }
  { assume (f-(K))◁pR
    with assms(1,2) have K◁Rt and K≠S
      using func1_1_L4 unfolding origin_ring.primeIdeal_def surj_def
      by auto
    moreover
    { fix J P assume jp: J∈target_ring.ideals
        P∈target_ring.ideals
        target_ring.productIdeal(J, P) ⊆ K
      from jp(3) have f-(target_ring.productIdeal(J, P)) ⊆ f-(K)
        by auto
      with assms(2) jp(1,2) have
        A: ((f-(J)) ·I (f-(P))) +I (ker) ⊆ f-(K)
        using prime_ideal_quot by auto
      from jp(1,2) have
        (f-(J))·I(f-(P)) ∪ ker ⊆ ((f-(J))·I(f-(P))) +I (ker)
        using preimage_ideal origin_ring.product_in_intersection(2)
          kernel_ideal origin_ring.comp_in_sum_ideals(3) by simp
      with A have ((f - J) ·I (f - P)) ⊆ f-(K) by auto
      with ‹(f-(K))◁pR› jp(1,2) have
        f-(J) ⊆ f-(K) ∨ f-(P) ⊆ f-(K)
        using preimage_ideal origin_ring.ideal_dest_subset
        unfolding origin_ring.primeIdeal_def by auto
      then have
        f(f-(J)) ⊆ f(f-(K)) ∨ f(f-(P)) ⊆ f(f-(K))
        by blast
      with assms jp(1,2) have J ⊆ K ∨ P ⊆ K
        using surj_image_vimage target_ring.ideal_dest_subset
        by auto
    }
    ultimately show K◁pRt
      unfolding target_ring.primeIdeal_def by auto
  }
qed
```

If the homomorphism is surjective then the function that maps ideals in the target ring to their inverse images (in the source ring) is a bijection between prime ideals in the target ring and the prime ideals containing the kernel in the source ring.

```
corollary (in ring_homo) bij_prime_ideals:
  defines F ≡ λJ∈target_ring.ideals. f-(J)
  assumes f ∈ surj(R,S)
  shows restrict(F,{J∈Pow(S). J◁pRt}) ∈
    bij({J∈Pow(S). J◁pRt}, {J∈Pow(R). ker⊆J ∧ (J◁pR)})
proof -
  let 𝓘t = target_ring.ideals
```

```
  from assms have I: F:ℐ_t → {K∈ℐ. ker ⊆ K}
    using ideal_quot_bijection bij_is_fun by simp
  have II: {J∈Pow(S). J◁_pR_t} ⊆ ℐ_t
    unfolding target_ring.primeIdeal_def by auto
  have III: {J∈Pow(S). J◁_pR_t} ⊆ ℐ_t
    unfolding target_ring.primeIdeal_def by auto
  with assms have restrict(F,{J∈Pow(S). J◁_pR_t}) ∈
    bij({J∈Pow(S). J◁_pR_t}, F{J∈Pow(S). J◁_pR_t})
    using restrict_bij ideal_quot_bijection unfolding bij_def
    by auto
  moreover have {t∈Pow(R). ker⊆t ∧ (t◁_pR)} = F{J∈Pow(S). J◁_pR_t}
  proof
    { fix t assume t ∈ F{J∈Pow(S). J◁_pR_t}
      with I III obtain q where q∈Pow(S) q◁_pR_t t=F(q)
        using func_imagedef by auto
      from ‹q∈Pow(S)› ‹q◁_pR_t› have q∈ℐ_t
        unfolding target_ring.primeIdeal_def by auto
      with I have F(q) ∈ {K∈ℐ. ker ⊆ K} using apply_funtype
        by blast
      with assms ‹q◁_pR_t› ‹t=F(q)› ‹q∈ℐ_t› ‹t=F(q)›
        have t◁R t◁_pR ker⊆t
        using prime_ideal_quot_3 by simp_all
    } then show F{J∈Pow(S). J◁_pR_t} ⊆ {t∈Pow(R). ker⊆t ∧ (t◁_pR)}
      using origin_ring.ideal_dest_subset by blast
    { fix t assume t∈{t∈Pow(R). ker⊆t ∧ (t◁_pR)}
      then have t∈Pow(R) ker⊆t t◁_pR by auto
      then have tSet: t ∈ {t∈ℐ. ker ⊆ t}
        unfolding origin_ring.primeIdeal_def by auto
      with assms have cont: converse(F)(t) ∈ ℐ_t
        using ideal_quot_bijection bij_converse_bij
          apply_funtype bij_is_fun by blast
      moreover from assms tSet have F(converse(F)(t)) = t
        using ideal_quot_bijection right_inverse_bij by blast
      ultimately have f-(converse(F)(t)) = t
        using assms(1) by simp
      with assms II tSet ‹t◁_pR› cont have t ∈ F{J∈Pow(S). J◁_pR_t}
        using prime_ideal_quot_3 target_ring.ideal_dest_subset
          bij_val_image_vimage ideal_quot_bijection
        unfolding target_ring.primeIdeal_def by auto
    } thus {t∈Pow(R). ker⊆t ∧ (t◁_pR)} ⊆ F{J∈Pow(S). J◁_pR_t}
      by auto
  qed
  ultimately show thesis by auto
qed

end
```

50 Rings - Commutative Rings

```
theory Ring_ZF_4 imports Ring_ZF_2 CommutativeSemigroup_ZF

begin

locale commutative_ring = ring0 +
  assumes commutative:M{is commutative on}R

lemma (in commutative_ring) mult_by_elem:
  assumes x∈R
  shows {x·y. y∈R}◁R
proof(rule ideal_intro)
  have x·1∈{x·y. y∈R} using Ring_ZF_1_L2(2) by auto
  then show {x·y. y∈R}≠0 by auto
  {
    fix t assume t:t∈{x · y . y ∈ R}
    then obtain y where y:y∈R t=x·y by auto
    then have t∈R using Ring_ZF_1_L4(3) assms by auto
  }
  then show {x · y . y ∈ R} ⊆ R by auto
  {
    fix t h assume ty:t∈{x · y . y ∈ R} h∈R
    from ty(1) obtain y where y:y:R t=x·y by auto
    from y(2) have t·h=(x·y)·h by auto
    then have t·h= x·(y·h) using Ring_ZF_1_L11(2)
      assms y(1) ty(2) by auto
    moreover have y·h∈R using y(1) ty(2) using Ring_ZF_1_L4(3) by auto
    ultimately have E1:t·h∈{x · y . y ∈ R} by auto
  }
  then show ∀t∈{x · y . y ∈ R}. ∀h∈R. t · h ∈ {x · y . y ∈ R} by auto
  {
    fix t h assume ty:t∈{x · y . y ∈ R} h∈R
    from ty(1) obtain y where y:y:R t=x·y by auto
    from y(2) have h·t = h·(x·y) by auto
    then have h·t = h·(y·x) using commutative
      unfolding IsCommutative_def using assms y(1) by auto
    then have h·t = (h·y)·x using Ring_ZF_1_L11(2)
      assms y(1) ty(2) by auto
    moreover have h·y∈R using y(1) ty(2) using Ring_ZF_1_L4(3) by auto
    ultimately have h·t = x·(h·y) h·y∈R using commutative
      unfolding IsCommutative_def using assms by auto
    then have h·t∈{x · y . y ∈ R} by auto
  }
  then show  ∀t∈{x · y . y ∈ R}. ∀h∈R. h · t ∈ {x · y . y ∈ R} by auto
  {
    fix t h assume th:t:{x · y . y ∈ R} h:{x · y . y ∈ R}
    then obtain yt yh where y:t=x·yt yt:R h=x·yh yh:R by auto
    then have t+h = (x·yt)+(x·yh) by auto
```

```
    then have t+h = x·(yt+yh) using ring_oper_distr(1) assms
      y(2,4) by auto
    moreover have yt+yh :R using y(2,4) Ring_ZF_1_L4(1) by auto
    ultimately have t+h:{x · y . y ∈ R} by auto
  }
  then show ∀xa∈{x · y . y ∈ R}. ∀y∈{x · y . y ∈ R}. xa + y ∈ {x · y
. y ∈ R} by auto
qed
```

```
theorem (in commutative_ring) principal_ideal:
  assumes x∈R
  shows ⟨{x}⟩_I = {x · y . y ∈ R}
proof
  have x·1∈{x·y. y∈R} using Ring_ZF_1_L2(2) by auto
  then have xx:x∈{x·y. y∈R} using Ring_ZF_1_L3(5) assms by auto
  then show ⟨{x}⟩_I ⊆ {x · y . y ∈ R}
    using generated_ideal_small mult_by_elem assms by auto
  {
    fix J assume j:{x} ⊆ J J◁R J∈Pow(R)
    {
      fix t assume t:t∈{x · y . y ∈ R}
      then obtain y where y:y:R t=x·y by auto
      with j have t∈J using ideal_dest_mult(1) by auto
    }
    then have {x · y . y ∈ R} ⊆ J by auto
  } moreover
  have {x · y . y ∈ R}∈{J∈Pow(R). J ◁R ∧ {x}⊆J}
    using assms Ring_ZF_1_L4(3) xx mult_by_elem by auto
  then have {J∈Pow(R). J ◁R ∧ {x}⊆J}≠0 by auto
  ultimately have {x · y . y ∈ R} ⊆ ⋂{J∈Pow(R). J ◁R ∧ {x}⊆J} by auto
  then show {x · y . y ∈ R} ⊆ ⟨{x}⟩_I
    using generatedIdeal_def assms by auto
qed
```

Commutative prime rings are the same as commutative ring with no zero divisors.

```
lemma (in commutative_ring) prime_ring_zero_divs_1:
  assumes [R,A,M]{is a prime ring}
  shows HasNoZeroDivs(R,A,M) unfolding HasNoZeroDivs_def
proof(safe)
  fix aa b assume as:aa∈R b∈R M ⟨aa, b⟩ =TheNeutralElement(R,A) b≠TheNeutralElement(R,A)
  with assms have (∀z∈R. M ⟨M ⟨aa, z⟩, b⟩ = 0) ⟶
                (aa = 0 ∨ b = 0) unfolding primeRing_def[OF ringAssum]
by auto
  moreover
  {
    fix z assume z:z∈R
    have (aa·z)·b = (z·aa)·b using as(1) z(1)
      commutative unfolding IsCommutative_def by auto
```

```
    then have (aa·z)·b = z·(aa·b) using Ring_ZF_1_L11(2)
      as z by auto
    then have (aa·z)·b = z·0 using as(3) by auto
    then have (aa·z)·b = 0 using Ring_ZF_1_L6(2) z by auto
  }
  ultimately have aa = 0 ∨ b = 0 by auto
  with as(4) show aa=TheNeutralElement(R,A) by auto
qed

lemma (in commutative_ring) prime_ring_zero_divs_2:
  assumes HasNoZeroDivs(R,A,M)
  shows [R,A,M]{is a prime ring} unfolding primeRing_def[OF ringAssum]
proof(safe)
  fix aa b assume as:aa∈R b∈R ∀z∈R. M  ⟨M  ⟨aa, z⟩, b⟩ = TheNeutralElement(R,
A)
    b≠TheNeutralElement(R,A)
  with assms have ∀c∈R. ∀b∈R. M  ⟨c, b⟩ = 0 ⟶
                c = 0 ∨ b = 0 unfolding HasNoZeroDivs_def by auto
  with as(1,2) have M  ⟨aa, b⟩ = 0 ⟶ aa = 0 ∨ b = 0 by auto
  with as(4) have M  ⟨aa, b⟩ = 0 ⟶ aa = 0 by auto
  moreover
  from as(3) have M⟨M  ⟨aa, 1⟩, b⟩ = 0 using Ring_ZF_1_L2(2) by auto
  then have aa·b = 0 using Ring_ZF_1_L3(5) as(1) by auto
  ultimately show aa = TheNeutralElement(R, A) by auto
qed

theorem (in ring0) prime_ideal_no_zero_divs:
  assumes I◁pR
  shows [QuotientBy(I),QuotientGroupOp(R, A, I),ProjFun2(R, QuotientGroupRel(R,
A, I), M)]{is a prime ring}
proof-
  have ideal:I◁R using assms unfolding primeIdeal_def by auto
  from primeRing_def[OF quotientBy_is_ring[of I]] assms have
    (∀x∈QuotientBy(I).
       ∀y∈QuotientBy(I).
          (∀z∈QuotientBy(I).
              ProjFun2(R, QuotientGroupRel(R, A, I), M)
              ⟨ProjFun2(R, QuotientGroupRel(R, A, I), M)  ⟨x, z⟩, y⟩ =
              TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I)))
⟶
          x = TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I))
∨
          y = TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I)))
⟶
[QuotientBy
      (I),QuotientGroupOp
           (R, A, I),ProjFun2(R, QuotientGroupRel(R, A, I), M)]{is a prime
ring}
    unfolding primeIdeal_def by auto moreover
```

```
  have ∀x∈QuotientBy(I).
      ∀y∈QuotientBy(I).
         (∀z∈QuotientBy(I).
             ProjFun2(R, QuotientGroupRel(R, A, I), M)
             ⟨ProjFun2(R, QuotientGroupRel(R, A, I), M)  ⟨x, z⟩, y⟩ =
             TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I)))
⟶
         x = TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I))
∨
         y = TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I))
  proof(safe)
    fix x y assume as:x∈QuotientBy(I) y∈QuotientBy(I) ∀z∈QuotientBy(I).
             ProjFun2(R, QuotientGroupRel(R, A, I), M)
             ⟨ProjFun2(R, QuotientGroupRel(R, A, I), M)  ⟨x, z⟩, y⟩ =
             TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I))
      y ≠ TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I))
    {
      fix xx assume xx:xx∈R QuotientGroupRel(R, A, I){xx} = x
      {
        fix yy assume yy:yy∈R QuotientGroupRel(R, A, I){yy} = y
        {
          fix zz assume zz:zz∈R
          have QuotientGroupRel(R, A, I){xx·zz·yy} =
            ProjFun2(R, QuotientGroupRel(R, A, I), M)⟨QuotientGroupRel(R,
A, I){xx·zz},QuotientGroupRel(R, A, I){yy}⟩
            using EquivClass_1_L10[OF ideal_equiv_rel[OF ideal] quotientBy_mul_monoid(1)[OF
ideal]]
            xx(1) yy(1) zz Ring_ZF_1_L4(3) by auto
          then have QuotientGroupRel(R, A, I){xx·zz·yy} =
            ProjFun2(R, QuotientGroupRel(R, A, I), M)⟨ProjFun2(R, QuotientGroupRel(R,
A, I), M)⟨QuotientGroupRel(R, A, I){xx},QuotientGroupRel(R, A, I){zz}⟩,QuotientGroupRel(R,
A, I){yy}⟩
            using EquivClass_1_L10[OF ideal_equiv_rel[OF ideal] quotientBy_mul_monoid(1)[OF
ideal]]
            xx(1) zz by auto
          with xx(2) yy(2) have QuotientGroupRel(R, A, I){xx·zz·yy} =

            ProjFun2(R, QuotientGroupRel(R, A, I), M)⟨ProjFun2(R, QuotientGroupRel(R,
A, I), M)⟨x,QuotientGroupRel(R, A, I){zz}⟩,y⟩
            by auto
          moreover have QuotientGroupRel(R, A, I){zz}∈QuotientBy(I) un-
folding QuotientBy_def[OF ideal]
            quotient_def using zz by auto
          moreover note as(3) ultimately
          have QuotientGroupRel(R, A, I){xx·zz·yy} = TheNeutralElement(QuotientBy(I),
QuotientGroupOp(R, A, I))
            by auto
          then have xx·zz·yy∈I using add_group.Group_ZF_2_4_L5E[OF ideal_normal_add_subgroup[OF
ideal],
```

```
            of xx·zz·yy QuotientGroupRel(R, A, I) TheNeutralElement(QuotientBy(I),
QuotientGroupOp(R, A, I))]
            using xx(1) yy(1) zz Ring_ZF_1_L4(3) unfolding QuotientBy_def[OF
ideal] by auto
        }
        then have ∀zz∈R. xx·zz·yy∈I by auto
        then have D:xx:I∨yy:I using assms equivalent_prime_ideal
          xx(1) yy(1) by auto
        {
          assume yy∈I
          then have False using add_group.Group_ZF_2_4_L5E[OF ideal_normal_add_subgroup[OF
ideal], of yy QuotientGroupRel(R,A,I)] yy as(4)
            unfolding QuotientBy_def[OF ideal] by auto
        }
        with D have xx∈I by auto
      }
      with quotientE[of y R QuotientGroupRel(R,A,I) xx∈I] have xx∈I us-
ing as(2) unfolding QuotientBy_def[OF ideal] by auto
      then have QuotientGroupRel(R, A, I){xx} = TheNeutralElement(QuotientBy(I),
QuotientGroupOp(R, A, I))
        using add_group.Group_ZF_2_4_L5E[OF ideal_normal_add_subgroup[OF
ideal], of xx QuotientGroupRel(R,A,I)]
        unfolding QuotientBy_def[OF ideal] using xx(1) by auto
      with xx(2) have x = TheNeutralElement(QuotientBy(I), QuotientGroupOp(R,
A, I)) by auto
    }
    with quotientE[of x R QuotientGroupRel(R,A,I) x = TheNeutralElement(QuotientBy(I),
QuotientGroupOp(R, A, I))]
    show x = TheNeutralElement(QuotientBy(I), QuotientGroupOp(R, A, I))
using as(1) unfolding QuotientBy_def[OF ideal] by auto
  qed
  ultimately show thesis by auto
qed


end
```

51 Fields - introduction

theory Field_ZF **imports** Ring_ZF

begin

This theory covers basic facts about fields.

51.1 Definition and basic properties

In this section we define what is a field and list the basic properties of fields.

Field is a notrivial commutative ring such that all non-zero elements have an inverse. We define the notion of being a field as a statement about three sets. The first set, denoted `K` is the carrier of the field. The second set, denoted `A` represents the additive operation on `K` (recall that in ZF set theory functions are sets). The third set `M` represents the multiplicative operation on `K`.

```
definition
  IsAfield(K,A,M) ≡
  (IsAring(K,A,M) ∧ (M {is commutative on} K) ∧
  TheNeutralElement(K,A) ≠ TheNeutralElement(K,M) ∧
  (∀a∈K. a≠TheNeutralElement(K,A)⟶
  (∃b∈K. M⟨a,b⟩ = TheNeutralElement(K,M))))
```

The `field0` context extends the `ring0` context adding field-related assumptions and notation related to the multiplicative inverse.

```
locale field0 = ring0 K A M for K A M +
  assumes mult_commute: M {is commutative on} K

  assumes not_triv: 0 ≠ 1

  assumes inv_exists: ∀x∈K. x≠0 ⟶ (∃y∈K. x·y = 1)

  fixes non_zero (K₀)
  defines non_zero_def[simp]: K₀ ≡ K-{0}

  fixes inv (_⁻¹  [96] 97)
  defines inv_def[simp]: a⁻¹ ≡ GroupInv(K₀,restrict(M,K₀×K₀))(a)
```

The next lemma assures us that we are talking fields in the `field0` context.

```
lemma (in field0) Field_ZF_1_L1: shows IsAfield(K,A,M)
  using ringAssum mult_commute not_triv inv_exists IsAfield_def
  by simp
```

We can use theorems proven in the `field0` context whenever we talk about a field.

```
lemma field_field0: assumes IsAfield(K,A,M)
  shows field0(K,A,M)
  using assms IsAfield_def field0_axioms.intro ring0_def field0_def
  by simp
```

Let's have an explicit statement that the multiplication in fields is commutative.

```
lemma (in field0) field_mult_comm: assumes a∈K  b∈K
  shows a·b = b·a
  using mult_commute assms IsCommutative_def by simp
```

Fields do not have zero divisors.

```
lemma (in field0) field_has_no_zero_divs: shows HasNoZeroDivs(K,A,M)
```

```
proof -
  { fix a b assume A1: a∈K  b∈K and A2: a·b = 0  and A3: b≠0
    from inv_exists A1 A3 obtain c where I: c∈K and II: b·c = 1
      by auto
    from A2 have a·b·c = 0·c by simp
    with A1 I have a·(b·c) = 0
      using Ring_ZF_1_L11 Ring_ZF_1_L6 by simp
    with A1 II have a=0 using Ring_ZF_1_L3 by simp }
  then have ∀a∈K.∀b∈K. a·b = 0 ⟶ a=0 ∨ b=0 by auto
    then show thesis using HasNoZeroDivs_def by auto
qed
```

K_0 (the set of nonzero field elements is closed with respect to multiplication.

```
lemma (in field0) Field_ZF_1_L2:
  shows K₀ {is closed under} M
  using Ring_ZF_1_L4 field_has_no_zero_divs Ring_ZF_1_L12
    IsOpClosed_def by auto
```

Any nonzero element has a right inverse that is nonzero.

```
lemma (in field0) Field_ZF_1_L3: assumes A1: a∈K₀
  shows ∃b∈K₀. a·b = 1
proof -
  from inv_exists A1 obtain b where b∈K and a·b = 1
    by auto
  with not_triv A1 show ∃b∈K₀. a·b = 1
    using Ring_ZF_1_L6 by auto
qed
```

If we remove zero, the field with multiplication becomes a group and we can use all theorems proven in `group0` context.

```
theorem (in field0) Field_ZF_1_L4: shows
  IsAgroup(K₀,restrict(M,K₀×K₀))
  group0(K₀,restrict(M,K₀×K₀))
  1 = TheNeutralElement(K₀,restrict(M,K₀×K₀))
proof-
  let f = restrict(M,K₀×K₀)
  have
    M {is associative on} K
    K₀ ⊆ K  K₀ {is closed under} M
    using Field_ZF_1_L1 IsAfield_def IsAring_def IsAgroup_def
      IsAmonoid_def Field_ZF_1_L2 by auto
  then have f {is associative on} K₀
    using func_ZF_4_L3 by simp
  moreover
  from not_triv have
    I: 1∈K₀ ∧ (∀a∈K₀. f⟨1,a⟩ = a ∧  f⟨a,1⟩ = a)
    using Ring_ZF_1_L2 Ring_ZF_1_L3 by auto
  then have ∃n∈K₀. ∀a∈K₀. f⟨n,a⟩ = a ∧  f⟨a,n⟩ = a
    by blast
```

```
  ultimately have II: IsAmonoid(K₀,f) using IsAmonoid_def
    by simp
  then have monoid0(K₀,f) using monoid0_def by simp
  moreover note I
  ultimately show 1 = TheNeutralElement(K₀,f)
    by (rule monoid0.group0_1_L4)
  then have ∀a∈K₀.∃b∈K₀. f⟨a,b⟩ =  TheNeutralElement(K₀,f)
    using Field_ZF_1_L3 by auto
  with II show IsAgroup(K₀,f) by (rule definition_of_group)
  then show group0(K₀,f) using group0_def by simp
qed
```

The inverse of a nonzero field element is nonzero.

```
lemma (in field0) Field_ZF_1_L5: assumes A1: a∈K  a≠0
  shows a⁻¹ ∈ K₀  (a⁻¹)² ∈ K₀   a⁻¹ ∈ K  a⁻¹ ≠ 0
proof -
  from A1 have a ∈ K₀ by simp
  then show a⁻¹ ∈ K₀ using Field_ZF_1_L4 group0.inverse_in_group
    by auto
  then show  (a⁻¹)² ∈ K₀  a⁻¹ ∈ K  a⁻¹ ≠ 0
    using Field_ZF_1_L2 IsOpClosed_def by auto
qed
```

The inverse is really the inverse.

```
lemma (in field0) Field_ZF_1_L6: assumes A1: a∈K  a≠0
  shows a·a⁻¹ = 1  a⁻¹·a = 1
proof -
  let f = restrict(M,K₀×K₀)
  from A1 have
    group0(K₀,f)
    a ∈ K₀
    using Field_ZF_1_L4 by auto
  then have
    f⟨a,GroupInv(K₀, f)(a)⟩ = TheNeutralElement(K₀,f) ∧
    f⟨GroupInv(K₀,f)(a),a⟩ = TheNeutralElement(K₀, f)
    by (rule group0.group0_2_L6)
  with A1 show a·a⁻¹ = 1  a⁻¹·a = 1
    using Field_ZF_1_L5 Field_ZF_1_L4 by auto
qed
```

A lemma with two field elements and cancelling.

```
lemma (in field0) Field_ZF_1_L7: assumes a∈K b∈K b≠0
  shows
  a·b·b⁻¹ = a
  a·b⁻¹·b = a
  using assms Field_ZF_1_L5 Ring_ZF_1_L11 Field_ZF_1_L6 Ring_ZF_1_L3
  by auto
```

51.2 Equations and identities

This section deals with more specialized identities that are true in fields.

$a/(a^2) = 1/a$.

```
lemma (in field0) Field_ZF_2_L1: assumes A1: a∈K  a≠0
  shows a·(a⁻¹)² = a⁻¹
proof -
  have a·(a⁻¹)² = a·(a⁻¹·a⁻¹) by simp
  also from A1 have ... =  (a·a⁻¹)·a⁻¹
    using Field_ZF_1_L5 Ring_ZF_1_L11
    by simp
  also from A1 have ... = a⁻¹
    using Field_ZF_1_L6 Field_ZF_1_L5 Ring_ZF_1_L3
    by simp
  finally show a·(a⁻¹)² = a⁻¹ by simp
qed
```

If we multiply two different numbers by a nonzero number, the results will be different.

```
lemma (in field0) Field_ZF_2_L2:
  assumes a∈K  b∈K  c∈K  a≠b  c≠0
  shows a·c⁻¹ ≠ b·c⁻¹
  using assms field_has_no_zero_divs Field_ZF_1_L5 Ring_ZF_1_L12B
  by simp
```

We can put a nonzero factor on the other side of non-identity (is this the best way to call it?) changing it to the inverse.

```
lemma (in field0) Field_ZF_2_L3:
  assumes A1: a∈K  b∈K  b≠0  c∈K   and A2: a·b ≠ c
  shows a ≠ c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ ≠ c·b⁻¹
    using  Ring_ZF_1_L4 Field_ZF_2_L2 by simp
  with A1 show a ≠ c·b⁻¹ using Field_ZF_1_L7
    by simp
qed
```

If if the inverse of b is different than a, then the inverse of a is different than b.

```
lemma (in field0) Field_ZF_2_L4:
  assumes a∈K  a≠0 and b⁻¹ ≠ a
  shows a⁻¹ ≠ b
  using assms Field_ZF_1_L4 group0.group0_2_L11B
  by simp
```

An identity with two field elements, one and an inverse.

```
lemma (in field0) Field_ZF_2_L5:
```

```
  assumes a∈K  b∈K b≠0
  shows (1 + a·b)·b⁻¹ = a + b⁻¹
  using assms Ring_ZF_1_L4 Field_ZF_1_L5 Ring_ZF_1_L2 ring_oper_distr

    Field_ZF_1_L7 Ring_ZF_1_L3 by simp
```

An identity with three field elements, inverse and cancelling.

```
lemma (in field0) Field_ZF_2_L6: assumes A1: a∈K  b∈K  b≠0  c∈K
  shows a·b·(c·b⁻¹) = a·c
proof -
  from A1 have T: a·b ∈ K  b⁻¹ ∈ K
    using Ring_ZF_1_L4 Field_ZF_1_L5 by auto
  with mult_commute A1 have a·b·(c·b⁻¹) = a·b·(b⁻¹·c)
    using IsCommutative_def by simp
  moreover
  from A1 T have a·b ∈ K  b⁻¹ ∈ K  c∈K
    by auto
  then have a·b·b⁻¹·c = a·b·(b⁻¹·c)
    by (rule Ring_ZF_1_L11)
  ultimately have a·b·(c·b⁻¹) = a·b·b⁻¹·c by simp
  with A1 show a·b·(c·b⁻¹) = a·c
    using Field_ZF_1_L7 by simp
qed
```

Inverse of an inverse of a non-zero element is the element.

```
lemma (in field0) non_zero_inv_inv: assumes a∈K a≠0
  shows (a⁻¹)⁻¹ = a
  using assms Field_ZF_1_L4(2) group0.group_inv_of_inv by simp
```

51.3 1/0=0

In ZF if $f : X \to Y$ and $x \notin X$ we have $f(x) = \emptyset$. Since $\emptyset$ (the empty set) in ZF is the same as zero of natural numbers we can claim that $1/0 = 0$ in certain sense. In this section we prove a theorem that makes makes it explicit.

The next locale extends the `field0` locale to introduce notation for division operation.

```
locale fieldd = field0 +
  fixes division
  defines division_def[simp]: division ≡ {⟨p,fst(p)·snd(p)⁻¹⟩. p∈K×K₀}

  fixes fdiv (infixl / 95)
  defines fdiv_def[simp]: x/y ≡ division⟨x,y⟩
```

Division is a function on $K \times K_0$ with values in K.

```
lemma (in fieldd) div_fun: shows division: K×K₀ → K
proof -
```

```
  have ∀p ∈ K×K0. fst(p)·snd(p)^-1 ∈ K
  proof
    fix p assume p ∈ K×K0
    hence fst(p) ∈ K and snd(p) ∈ K0 by auto
    then show  fst(p)·snd(p)^-1 ∈ K using Ring_ZF_1_L4 Field_ZF_1_L5 by
auto
  qed
  then have {⟨p,fst(p)·snd(p)^-1⟩. p∈K×K0}: K×K0 → K
    by (rule ZF_fun_from_total)
  thus thesis by simp
qed
```

So, really $1/0 = 0$. The essential lemma is `apply_0` from standard Isabelle's `func.thy`.

```
theorem (in fieldd) one_over_zero: shows 1/0 = 0
proof-
  have domain(division) = K×K0 using div_fun func1_1_L1
    by simp
  hence ⟨1,0⟩ ∉ domain(division) by auto
  then show thesis using apply_0 by simp
qed

end
```

52 Modules

```
theory Module_ZF imports Ring_ZF_3 Field_ZF

begin
```

A module is a generalization of the concept of a vector space in which scalars do not form a field but a ring.

52.1 Definition and basic properties of modules

Let R be a ring and M be an abelian group. The most common definition of a left R-module posits the existence of a scalar multiplication operation $R \times M \to M$ satisfying certain four properties. Here we take a bit more concise and abstract approach defining a module as a ring action on an abelian group.

We know that endomorphisms of an abelian group $\mathcal{M}$ form a ring with pointwise addition as the additive operation and composition as the ring multiplication. This asssertion is a bit imprecise though as pointwise addition is a binary operation on the space of functions $\mathcal{M} \to \mathcal{M}$ (i.e. its domain is $(\mathcal{M} \to \mathcal{M}) \times (\mathcal{M} \to \mathcal{M})$) while we need the space of endomorphisms to be the domain of the ring addition and multiplication. Therefore, to get the actual additive operation we need to restrict the pointwise addition of functions $\mathcal{M} \rightarrow \mathcal{M}$ to the set of endomorphisms of $\mathcal{M}$. Recall from the `Group_ZF_5` that the `InEnd` operator restricts an operation to the set of endomorphisms and see the `func_ZF` theory for definitions of lifting an operation on a set to a function space over that set.

definition `EndAdd(`$\mathcal{M}$`,A) ≡ InEnd(A {lifted to function space over}` $\mathcal{M}$`,`$\mathcal{M}$`,A)`

Similarly we define the multiplication in the ring of endomorphisms as the restriction of compositions to the endomorphisms of $\mathcal{M}$. See the `func_ZF` theory for the definition of the `Composition` operator.

definition `EndMult(`$\mathcal{M}$`,A) ≡ InEnd(Composition(`$\mathcal{M}$`),`$\mathcal{M}$`,A)`

We can now reformulate the theorem `end_is_ring` from the `Group_ZF_5` theory in terms of the addition and multiplication of endomorphisms defined above.

lemma (in `abelian_group) end_is_ring1:`
  shows `IsAring(End(G,P),EndAdd(G,P),EndMult(G,P))`
  using `end_is_ring` **unfolding** `EndAdd_def EndMult_def` **by** `simp`

We define an `action` as a homomorphism into a space of endomorphisms (typically of some abelian group). In the definition below S is the set of scalars, A is the addition operation on this set, M is multiplication on the set, $\mathcal{M}$ is the group, A_M is the group operation, and H is the ring homomorphism of the ring of scalars to the ring of endomorphisms of the group. On the right hand side of the definition `End(`$\mathcal{M}$`,A`$_V$`)` is the set of endomorphisms of the group $\mathcal{M}$ with operation A_M. This definition is only ever used as part of the definition of a module and vector space, it's just convenient to split it off to shorten the main definitions.

definition
  `IsAction(S,A,M,`$\mathcal{M}$`,A`$_M$`,H) ≡ ringHomomor(H,S,A,M,End(`$\mathcal{M}$`,A`$_M$`),EndAdd(`$\mathcal{M}$`,A`$_M$`),EndMult(`$\mathcal{M}$`,A`$_M$`))`

A module is a ring action on an abelian group.

definition `IsLeftModule(S,A,M,`$\mathcal{M}$`,A`$_M$`,H) ≡`
  `IsAring(S,A,M) ∧ IsAgroup(`$\mathcal{M}$`,A`$_M$`) ∧ (A`$_M$` {is commutative on}` $\mathcal{M}$`) ∧`
`IsAction(S,A,M,`$\mathcal{M}$`,A`$_M$`,H)`

If H defines a module then it is a ring action, hence a ring homomorphism, hence a function on that ring.

lemma `module_action_type:` **assumes** `IsLeftModule(S,A,M,`$\mathcal{M}$`,A`$_M$`,H)`
  shows
    `IsAction(S,A,M,`$\mathcal{M}$`,A`$_M$`,H)`
    `ringHomomor(H,S,A,M,End(`$\mathcal{M}$`,A`$_M$`),EndAdd(`$\mathcal{M}$`,A`$_M$`),EndMult(`$\mathcal{M}$`,A`$_M$`))`
    `H:S→End(`$\mathcal{M}$`,A`$_M$`)`
  using `assms` **unfolding** `IsLeftModule_def IsAction_def ringHomomor_def`
by `auto`

The next locale defines context (i.e. common assumptions and notation) when considering modules. We reuse notation from the `ring0` locale and add notation specific to modules. The addition and multiplication in the ring of scalars is denoted $+$ and $\cdot$, resp. The addition of module elements will be denoted $+_V$. The multiplication (scaling) of scalars by module elemenst will be denoted $\cdot_S$. Θ is the zero module element, i.e. the neutral element of the abelian group of the module elements.

locale module0 = ring0 +

  fixes $\mathcal{M}$ A_M H

  assumes mAbGr: IsAgroup($\mathcal{M}$,A_M) $\wedge$ (A_M {is commutative on} $\mathcal{M}$)

  assumes mAction: IsAction(R,A,M,$\mathcal{M}$,A_M,H)

  fixes zero_vec (Θ)
  defines zero_vec_def [simp]: $\Theta \equiv$ TheNeutralElement($\mathcal{M}$,A_M)

  fixes vAdd (**infixl** $+_V$ 80)
  defines vAdd_def [simp]: $v_1 +_V v_2 \equiv A_M\langle v_1,v_2\rangle$

  fixes scal (**infix** $\cdot_S$ 90)
  defines scal_def [simp]: s $\cdot_S$ v $\equiv$ (H(s))(v)

  fixes negV ($-$_)
  defines negV_def [simp]: $-$v $\equiv$ GroupInv($\mathcal{M}$,A_M)(v)

  fixes vSub (**infix** $-_V$ 80)
  defines vSub_def [simp]: $v_1 -_V v_2 \equiv v_1 +_V (-v_2)$

We indeed talk about modules in the `module0` context.

lemma (**in** module0) module_in_module0: **shows** IsLeftModule(R,A,M,$\mathcal{M}$,A_M,H)
  using ringAssum mAbGr mAction **unfolding** IsLeftModule_def **by** simp

Theorems proven in the `abelian_group` context are valid as applied to the `module0` context as applied to the abelian group of module elements.

lemma (**in** module0) abelian_group_valid_module0:
  shows abelian_group($\mathcal{M}$,A_M)
  using mAbGr group0_def abelian_group_def abelian_group_axioms_def
  by simp

Another way to state that theorems proven in the `abelian_group` context can be used in the `module0` context:

sublocale module0 < mod_ab_gr: abelian_group $\mathcal{M}$ A_M Θ vAdd negV
  λs. Fold(A_M, Θ, s) λ n x. Fold(A_M, Θ, {$\langle$k, x$\rangle$. k $\in$ n})
  using abelian_group_valid_module0 **by** auto

Theorems proven in the `ring_homo` context are valid in the `module0` context, as applied to ring R and the ring of endomorphisms of the group of module elements.

```
lemma (in module0) ring_homo_valid_module0:
  shows ring_homo(R,A,M,End(𝓜,A_M),EndAdd(𝓜,A_M),EndMult(𝓜,A_M),H)
  using ringAssum mAction abelian_group_valid_module0 abelian_group.end_is_ring1
  unfolding IsAction_def ring_homo_def by simp
```

Another way to make theorems proven in the `ring_homo` context available in the `module0` context:

```
sublocale module0 < vec_act_homo: ring_homo R A M
  End(𝓜,A_M) EndAdd(𝓜,A_M) EndMult(𝓜,A_M) H
  ringa
  ringminus
  ringsub
  ringm
  ringzero
  ringone
  ringtwo
  ringsq
  λx y.  EndAdd(𝓜,A_M)  ⟨x, y⟩
  λx. GroupInv(End(𝓜,A_M), EndAdd(𝓜, A_M))(x)
  λx y. EndAdd(𝓜,A_M)⟨x,GroupInv(End(𝓜, A_M), EndAdd(𝓜, A_M))(y)⟩
  λx y. EndMult(𝓜,A_M)⟨x, y⟩
  TheNeutralElement(End(𝓜, A_M), EndAdd(𝓜, A_M))
  TheNeutralElement(End(𝓜, A_M), EndMult(𝓜, A_M))
  EndAdd(𝓜,A_M)⟨TheNeutralElement(End(𝓜, A_M),EndMult(𝓜, A_M)),TheNeutralElement(End(𝓜,
A_M), EndMult(𝓜,A_M))⟩
  λx.  EndMult(𝓜,A_M)⟨x, x⟩
  using ring_homo_valid_module0 by auto
```

In the ring of endomorphisms of the module the neutral element of the the multiplicative operation is the identity function. The neutral element of the additive operation is the zero valued constant function, which is also the value of the homomorphism that defines the module at zero.

```
lemma (in module0) add_mult_neut_elems: shows
  TheNeutralElement(End(𝓜, A_M), EndMult(𝓜,A_M)) = id(𝓜) and
  TheNeutralElement(End(𝓜,A_M),EndAdd(𝓜,A_M)) = ConstantFunction(𝓜,Θ)
  H(0) = ConstantFunction(𝓜,Θ)
proof -
  show TheNeutralElement(End(𝓜, A_M), EndMult(𝓜, A_M)) = id(𝓜)
    using mod_ab_gr.end_comp_monoid(2) unfolding EndMult_def
    by blast
  have H(0) = TheNeutralElement(End(𝓜,A_M),EndAdd(𝓜,A_M))
    using vec_act_homo.homomor_dest_zero by blast
  also show
    TheNeutralElement(End(𝓜, A_M), EndAdd(𝓜, A_M)) = ConstantFunction(𝓜,Θ)
    using mod_ab_gr.end_add_neut_elem unfolding EndAdd_def by blast
```

```
  finally show H(0) = ConstantFunction(ℳ,Θ) by simp
qed
```

The value of the homomorphism defining the module is an endomorphism of the group of module elements and hence a function that maps the module into itself.

```
lemma (in module0) H_val_type: assumes r∈R shows
  H(r) ∈ End(ℳ,A_M) and H(r):ℳ→ℳ
  using mAction assms apply_funtype unfolding IsAction_def ringHomomor_def
End_def
  by auto
```

In the `module0` context the neutral element of addition of module elements is denoted Θ. Of course Θ is an element of the module.

```
lemma (in module0) zero_in_mod: shows Θ ∈ ℳ
  using mod_ab_gr.group0_2_L2 by simp
```

Θ is indeed the neutral element of addition of module elements.

```
lemma (in module0) zero_neutral: assumes x∈ℳ
  shows x +_V Θ = x and Θ +_V x = x
  using assms  mod_ab_gr.group0_2_L2 by simp_all
```

52.2 Module axioms

A more common definition of a module assumes that R is a ring, V is an abelian group and lists a couple of properties that the multiplications of scalars (elements of R) by the elements of the module V should have. In this section we show that the definition of a module as a ring action on an abelian group V implies these properties.

Θ is fixed by scalar multiplication.

```
lemma (in module0) zero_fixed: assumes r∈R
  shows r ·_S Θ = Θ
  using mAbGr assms H_val_type(1) image_neutral unfolding End_def by auto
```

The scalar multiplication is distributive with respect to the module addition.

```
lemma (in module0) module_ax1: assumes r∈R x∈ℳ y∈ℳ
  shows r·_S(x+_Vy) = r·_Sx +_V r·_Sy
  using assms H_val_type(1) mod_ab_gr.endomor_eq by simp
```

The scalar addition is distributive with respect to scalar multiplication.

```
lemma (in module0) module_ax2: assumes r∈R s∈R x∈ℳ
  shows (r+s)·_Sx = r·_Sx +_V s·_Sx
  using assms vec_act_homo.homomor_dest_add H_val_type(1) mod_ab_gr.end_pointwise_add_val
  unfolding EndAdd_def by simp
```

Multiplication by scalars is associative with multiplication of scalars.

```
lemma (in module0) module_ax3: assumes r∈R s∈R x∈ℳ
  shows (r·s)·Sx = r·S(s·Sx)
proof -
  let e = EndMult(ℳ,AM)⟨H(r),H(s)⟩
  have (r·s)·Sx = (H(r·s))(x) by simp
  also have (H(r·s))(x) = e(x)
  proof -
    from mAction assms(1,2) have H(r·s) = e
      using vec_act_homo.homomor_dest_mult unfolding IsAction_def
      by blast
    then show thesis by (rule same_constr)
  qed
  also have  e(x) = r·S(s·Sx)
  proof -
    from assms(1,2) have e(x) = (Composition(ℳ)⟨H(r),H(s)⟩)(x)
      using H_val_type(1) unfolding EndMult_def by simp
    also from assms have ... =  r·S(s·Sx) using H_val_type(2) func_ZF_5_L3
      by simp
    finally show e(x) = r·S(s·Sx) by blast
  qed
  finally show thesis by simp
qed
```

Scaling a module element by one gives the same module element.

```
lemma (in module0) module_ax4: assumes x∈ℳ shows 1·Sx = x
proof -
  let n = TheNeutralElement(End(ℳ,AM),EndMult(ℳ,AM))
  from mAction have H(TheNeutralElement(R,M)) = n
    unfolding IsAction_def ringHomomor_def by blast
  moreover have TheNeutralElement(R,M) = 1 by simp
  ultimately have H(1) = n by blast
  also have n = id(ℳ) by (rule add_mult_neut_elems)
  finally have H(1) = id(ℳ) by simp
  with assms show 1·Sx = x by simp
qed
```

Multiplying by zero is zero.

```
lemma (in module0) mult_zero:
  assumes g∈ℳ shows 0·Sg = Θ
  using assms add_mult_neut_elems(3) func1_3_L2 by simp
```

Taking inverses in a module is just multiplying by -1

```
lemma (in module0) inv_module:
  assumes g∈ℳ
  shows (-1)·Sg = − g
proof-
  have 1∈R using Ring_ZF_1_L2(2) by auto
  then have (-1) ∈ R using Ring_ZF_1_L3(1) by auto
  with assms have g +V ((-1)·Sg) = (1·Sg) +V ((-1)·Sg)
```

```
    using module_ax4 by auto
  also from assms <1∈R> <(-1) ∈ R> have ... = (1-1)·_S g
    using module_ax2 unfolding ringsub_def by auto
  also from <1∈R> have ... = 0·_S g using Ring_ZF_1_L3(7) by auto
  also from assms have ... = Θ using mult_zero by auto
  finally have g +_V ((-1)·_S g) = Θ by auto
  moreover
  from <(-1) ∈ R> have H(-1)∈𝓜→𝓜 using H_val_type(2) by auto
  with assms have (-1)·_S g ∈ 𝓜 unfolding End_def using apply_type by
auto
  ultimately show thesis using assms mod_ab_gr.group0_2_L9(2) by auto
qed

end
```

52.3 Linear Combinations on Modules

```
theory Module_ZF_1 imports Module_ZF CommutativeSemigroup_ZF

begin
```

Since modules are abelian groups, we can make use of its commutativity to create new elements by adding acted on elements finitely. Consider two ordered collections of ring elements and of group elements (indexed by a finite set); then we can add their actions to obtain a new group element. This is a linear combination.

```
definition(in module0)
  LinearComb (∑ [_;{_,_}] 88)
  where fR:C→R ⟹ fG:C→𝓜 ⟹ D∈FinPow(C) ⟹ LinearComb(D,fR,fG)
≡ if D≠0 then CommSetFold(A_M,{⟨m,(fRm)·_S (fGm)⟩. m∈domain(fR)},D)
    else Θ
```

The function that for each index element gives us the acted element of the abelian group is a function from the index to the group.

```
lemma(in module0) coordinate_function:
  assumes AA:C→R B:C→𝓜
  shows {⟨m,(AAm)·_S (Bm)⟩. m∈C}:C→𝓜
proof-
  {
    fix m assume m∈C
    then have p:AAm∈RBm∈𝓜 using assms(1,2) apply_type by auto
    from p(1) have H(AAm):𝓜→𝓜 using H_val_type(2) by auto
    then have (AAm)·_S(Bm)∈𝓜 using p(2) apply_type by auto
  }
  then have {⟨m,(AAm)·_S (Bm)⟩. m∈C}⊆C×𝓜 by auto moreover
  {
    fix x y assume ⟨x,y⟩∈{⟨m,(AAm)·_S (Bm)⟩. m∈C}
    then have xx:x∈C y=(AAx)·_S (Bx) by auto
```

```
    {
      fix y' assume ⟨x,y'⟩∈{⟨m,(AAm)·S (Bm)⟩. m∈C}
      then have y'=(AAx)·S (Bx) by auto
      with xx(2) have y=y' by auto
    }
    then have ∀y'. ⟨x,y'⟩∈{⟨m,(AAm)·S (Bm)⟩. m∈C} ⟶ y=y' by auto
  }
  then have ∀x y. ⟨x,y⟩∈{⟨m,(AAm)·S (Bm)⟩. m∈C} ⟶ (∀y'. ⟨x,y'⟩∈{⟨m,(AAm)·S
(Bm)⟩. m∈C} ⟶ y=y')
    by auto
  moreover have domain({⟨m,(AAm)·S (Bm)⟩. m∈C})⊆C unfolding domain_def
by auto ultimately
  show {⟨m,(AAm)·S (Bm)⟩. m∈C}:C→𝓜 unfolding Pi_def function_def by
auto
qed
```

A linear combination results in a group element where the functions and the sets are well defined.

```
theorem(in module0) linComb_is_in_module:
  assumes AA:C→R B:C→𝓜 D∈FinPow(C)
  shows (∑[D;{AA,B}])∈𝓜
proof-
  {
    assume noE:D≠0
    have ffun:{⟨m,(AAm)·S (Bm)⟩. m∈C}:C→𝓜 using coordinate_function
assms by auto
    note commsemigr.sum_over_set_add_point(1)[OF
      commsemigr.intro[OF _ _ ffun] assms(3) noE]
      mod_ab_gr.abelian_group_axioms abelian_group_def
      group0_def IsAgroup_def IsAmonoid_def abelian_group_axioms_def more-
over
    from assms(1) have domain(AA)=C unfolding Pi_def by auto
    ultimately have thesis unfolding LinearComb_def[OF assms(1-3)] us-
ing noE by auto
  }
  then show thesis unfolding LinearComb_def[OF assms(1-3)]
    using mod_ab_gr.group0_2_L2 by auto
qed
```

A linear combination of one element functions is just the action of one element onto another.

```
lemma(in module0) linComb_one_element:
  assumes x∈X AA:X→R B:X→𝓜
  shows ∑[{x};{AA,B}]=(AAx)·S(Bx)
proof-
  have dom:domain(AA)=X using assms(2) func1_1_L1 by auto
  have fin:{x}∈FinPow(X) unfolding FinPow_def using assms(1) by auto
  have A:∑[{x};{AA,B}]=CommSetFold(AM,{⟨t,(AAt)·S (Bt)⟩. t∈X},{x})
    unfolding LinearComb_def[OF assms(2,3) fin] dom by auto
```

```
  have assoc:A_M{is associative on}M using mod_ab_gr.abelian_group_axioms
    unfolding abelian_group_def group0_def IsAgroup_def IsAmonoid_def
by auto
  have comm:commsemigr(M, A_M, X, {⟨t,(AAt)·_S (Bt)⟩. t∈X})
    unfolding commsemigr_def
    using mod_ab_gr.abelian_group_axioms
    unfolding abelian_group_def abelian_group_axioms_def
    group0_def IsAgroup_def IsAmonoid_def using coordinate_function[OF
assms(2,3)] by auto
  have sing:1≈{x} |{x}|=1 using singleton_eqpoll_1 eqpoll_sym by auto
  then obtain b where b:b∈bij(|{x}|,{x}) b∈bij(1,{x}) unfolding eqpoll_def
by auto then
  have ∑ [{x};{AA,B}]=Fold1(A_M,{⟨t,(AAt)·_S (Bt)⟩. t∈X} O b)
    using commsemigr.sum_over_set_bij[OF comm fin, of b] trans[OF A] by
blast
  also have ...=({⟨t,(AAt)·_S (Bt)⟩. t∈X} O b)0 using semigr0.prod_of_1elem
unfolding semigr0_def using
    comp_fun[OF _ coordinate_function[OF assms(2,3)], of b 1] b(2) assms(1)
func1_1_L1B[of b 1 {x} X] assoc unfolding bij_def inj_def by blast
  also have ...=({⟨t,(AAt)·_S (Bt)⟩. t∈X})(b0) using comp_fun_apply b un-
folding bij_def inj_def by auto
  also have ...= {⟨t,(AAt)·_S (Bt)⟩. t∈X}x using apply_type[of b 1 λt. {x}
0] b unfolding bij_def inj_def
    by auto
  ultimately show thesis using apply_equality[OF _ coordinate_function[OF
assms(2,3)]] assms(1) by auto
qed
```

Since a linear combination is a group element, it makes sense to apply the action onto it. With this result we simplify it to a linear combination.

```
lemma(in module0) linComb_action:
  assumes AA:X→R B:X→M r∈R D∈FinPow(X)
  shows r·_S(∑ [D;{AA,B}])=∑ [D;{{⟨k,r·(AAk)⟩. k∈X},B}]
    and {⟨m,r ·(AAm)⟩. m∈X}:X→R
proof-
  show f:{⟨m,r ·(AAm)⟩. m∈X}:X→R unfolding Pi_def function_def domain_def
    using apply_type[OF assms(1)] assms(3) Ring_ZF_1_L4(3) by auto
  {
    fix AAt Bt assume as:AAt:X → R Bt:X → M
    have ∑ [0;{AAt,Bt}]=Θ using LinearComb_def[OF as] unfolding FinPow_def
by auto
    then have r ·_S (∑ [0;{AAt,Bt}])=Θ using assms(3) zero_fixed by auto
moreover
    have mr:{⟨m,r·(AAtm)⟩. m∈X}:X→R unfolding Pi_def function_def domain_def
      using apply_type[OF as(1)] assms(3) Ring_ZF_1_L4(3) by auto
    then have ∑ [0;{{⟨x, r · (AAt  x)⟩ . x ∈ X},Bt}]=Θ using LinearComb_def[OF
mr as(2)]
      unfolding FinPow_def by auto
    ultimately have r ·_S (∑ [0;{AAt,Bt}]) = ∑ [0;{{⟨x, r ·(AAt  x)⟩ . x
```

```
∈ X},Bt}] by auto
  }
  then have case0:(∀AAt∈X → R.
    ∀Bt∈X → 𝓜. r ·_S (∑[0;{AAt,Bt}]) = ∑[0;{{⟨x, r · (AAt  x)⟩ . x
∈ X},Bt}]) by auto
  {
    fix Tk assume A:Tk∈FinPow(X) Tk≠0
    then obtain t where tt:t∈Tk by auto
    {
      assume (∀AAk∈X→R. ∀Bk∈X→𝓜. (r·_S(∑[Tk-{t};{AAk,Bk}])=∑[Tk-{t};{{⟨k,r·(AAkk)⟩.
k∈X},Bk}]))
      with tt obtain t where t:t∈Tk (∀AAk∈X→R. ∀Bk∈X→𝓜. (r·_S(∑[Tk-{t};{AAk,Bk}])=∑[T
k∈X},Bk}])) by auto
      let Tk=Tk-{t}
      have CC:Tk=Tk∪{t} using t by auto
      have DD:Tk∈FinPow(X) using A(1) unfolding FinPow_def using subset_Finite[of
Tk Tk]
        by auto
      have tX:t∈X using t(1) A(1) unfolding FinPow_def by auto
      then have EE:X-(Tk)=(X-Tk)∪{t} using t(1) by auto
      {
        assume as:Tk-{t}≠0
        with t have BB:t∈Tk Tk-{t}≠0 ∀AAk∈X→R. ∀Bk∈X→𝓜. (r·_S(∑[Tk-{t};{AAk,Bk}])=∑[T
k∈X},Bk}]) by auto
        from BB(3) have A3:∀AAk∈X→R. ∀Bk∈X→𝓜. (r·_S(∑[Tk;{AAk,Bk}])=∑[Tk;{{⟨k,r·(AAkk)
k∈X},Bk}]) by auto
        {
          fix AAt Bt assume B:AAt:X→R Bt:X→𝓜
          then have af:{⟨k,(AAtk)·_S(Btk)⟩. k∈X}:X→𝓜 using coordinate_function
by auto
          have comm:commsemigr(𝓜, A_M, X, {⟨k,(AAtk)·_S(Btk)⟩. k∈X})
            unfolding commsemigr_def
            using mod_ab_gr.abelian_group_axioms
            unfolding abelian_group_def abelian_group_axioms_def
            group0_def IsAgroup_def IsAmonoid_def using af by auto
          then have CommSetFold(A_M,{⟨k,(AAtk)·_S(Btk)⟩. k∈X},Tk)=CommSetFold(A_M,{⟨k,(AAtk)·_S
k∈X},Tk)
            +_V({⟨k,(AAtk)·_S(Btk)⟩. k∈X}t) using commsemigr.sum_over_set_add_point(2)[of
𝓜 A_M X {⟨k,(AAtk)·_S(Btk)⟩. k∈X} Tk-{t}] DD BB(2)
            A(1-2) EE CC by auto
          also have ...=CommSetFold(A_M,{⟨k,(AAtk)·_S(Btk)⟩. k∈X},Tk)
            +_V((AAtt)·_S(Btt)) using apply_equality[OF _ af, of t (AAtt)·_S(Btt)]
tX by auto
          ultimately have CommSetFold(A_M,{⟨k,(AAtk)·_S(Btk)⟩. k∈X},Tk)=CommSetFold(A_M,{⟨k,(A
k∈X},Tk)
            +_V((AAtt)·_S(Btt)) by auto moreover
          have domain(AAt)=X using B(1) Pi_def by auto ultimately
          have (∑[Tk;{AAt,Bt}])=(∑[Tk;{AAt,Bt}])+_V((AAtt)·_S(Btt)) un-
folding LinearComb_def[OF B(1,2) A(1)]
```

```
      LinearComb_def[OF B(1,2) DD] using as A(2) by auto
    then have eq:r·S(∑[Tk;{AAt,Bt}])=r·S((∑[Tk;{AAt,Bt}])+V((AAtt)·S(Btt)))
by auto
    moreover have ∀g∈𝓜. ∀h∈𝓜. (Hr)(g+Vh)=((Hr)g)+V((Hr)h)

      using module_ax1 assms(3) by auto
    moreover have AR:AAtt∈R using B(1) tX apply_type by auto
    then have H(AAtt):𝓜→𝓜 using H_val_type(2) by auto
    from apply_type[OF this] have (AAtt)·S(Btt)∈𝓜 using apply_type[OF
B(2)] tX by auto moreover
    have mr:{⟨m,r·(AAtm)⟩. m∈X}:X→R unfolding Pi_def function_def
domain_def
      using apply_type[OF B(1)] assms(3) Ring_ZF_1_L4(3) by auto
    then have fff:{⟨m,({⟨m,r·(AAtm)⟩. m∈X}m)·S(Btm)⟩. m∈X}:X→𝓜
using coordinate_function[OF mr B(2)] apply_equality[OF _ mr] by auto
    with tX have pff:⟨t,({⟨m,r·(AAtm)⟩. m∈X}t)·S(Btt)⟩∈{⟨m,({⟨m,r·(AAtm)⟩.
m∈X}m)·S(Btm)⟩. m∈X}
      by auto
    have dom:domain({⟨m,(r·(AAtm))⟩. m∈X})=X by auto
    have comm2:commsemigr(𝓜, AM, X, {⟨m, ({⟨x,r ·(AAtx)⟩. x∈X}m)·S(Btm)⟩.
m∈X})
      unfolding commsemigr_def
      using mod_ab_gr.abelian_group_axioms
      unfolding abelian_group_def abelian_group_axioms_def
      group0_def IsAgroup_def IsAmonoid_def using fff by auto
    have ∑[Tk;{AAt,Bt}]∈𝓜 using linComb_is_in_module[OF B(1,2)
DD].
    ultimately have r ·S ((∑[Tk;{AAt,Bt}]) +V ((AAt  t) ·S (Bt
t)))=(r·S(∑[Tk;{AAt,Bt}]))+V(r·S((AAtt)·S(Btt)))
      by auto
    also have ...=(r·S(∑[Tk;{AAt,Bt}]))+V((r·(AAtt))·S(Btt)) us-
ing module_ax3
      assms(3) AR apply_type[OF B(2)] tX by auto
    also have ...=(∑[Tk;{{⟨x,r·(AAtx)⟩. x∈X},Bt}])+V((r·(AAtt))·S(Btt))
      using A3 B(1,2) by auto
    also have ...=(∑[Tk;{{⟨x,r·(AAtx)⟩. x∈X},Bt}])+V(({⟨m, r · (AAt
m)⟩ . m ∈ X} t)·S(Btt))
      using apply_equality[OF _ mr] tX by auto
    also have ...=(∑[Tk;{{⟨x,r·(AAtx)⟩. x∈X},Bt}])+V({⟨m, ({⟨x,(r
·(AAtx))⟩. x∈X}m)·S(Btm)⟩. m∈X}t)
      using apply_equality[OF pff fff] by auto
    also have ...=CommSetFold(AM,{⟨m, ({⟨x,r ·(AAtx)⟩. x∈X}m)·S(Btm)⟩.
m∈X},Tk)+V({⟨m, ({⟨x,(r ·(AAtx))⟩. x∈X}m)·S(Btm)⟩. m∈X}t)
      unfolding LinearComb_def[OF mr B(2) DD] using dom as by auto
    also have ...=CommSetFold(AM,{⟨m, ({⟨x,r ·(AAtx)⟩. x∈X}m)·S(Btm)⟩.
m∈X},Tk) using
      commsemigr.sum_over_set_add_point(2)[OF comm2, of Tk-{t}]
fff DD tX CC BB(2) by auto
    ultimately have r ·S ((∑[Tk;{AAt,Bt}]) +V ((AAt  t) ·S (Bt
```

```
t))) =CommSetFold(A_M,{⟨m, ({⟨x,r ·(AAtx)⟩. x∈X}m)·_S(Btm)⟩. m∈X},Tk)
            by auto
          with eq have r ·_S (∑[Tk;{AAt,Bt}]) =CommSetFold(A_M,{⟨m, ({⟨x,r
·(AAtx)⟩. x∈X}m)·_S(Btm)⟩. m∈X},Tk) by auto
          also have ...=∑[Tk;{{⟨x,r·(AAtx)⟩. x∈X},Bt}] using A(2) unfolding LinearComb_def[OF mr B(2) A(1)] dom by auto
          ultimately have r ·_S (∑[Tk;{AAt,Bt}]) =∑[Tk;{{⟨x,r·(AAtx)⟩.
x∈X},Bt}] by auto
        }
        then have ∀AA∈X→R. ∀B∈X→ℳ. r ·_S (∑[Tk;{AA,B}]) =∑[Tk;{{⟨x,r·(AAx)⟩.
x∈X},B}] using BB by auto
      }
      moreover
      {
        assume Tk-{t}=0
        then have sing:Tk={t} using A(2) by auto
        {
          fix AAt Bt assume B:AAt:X→R Bt:X→ℳ
          have mr:{⟨m,r·(AAtm)⟩. m∈X}:X→R unfolding Pi_def function_def
domain_def
            using apply_type[OF B(1)] assms(3) Ring_ZF_1_L4(3) by auto
          from sing have ∑[Tk;{AAt,Bt}]=(AAtt)·_S(Btt) using linComb_one_element[OF
tX B]
            by auto
          then have r ·_S (∑[Tk;{AAt,Bt}])=r ·_S ((AAtt)·_S(Btt)) by auto
          also have ...=(r · (AAtt))·_S(Btt) using module_ax3 assms(3) apply_type[OF
B(1) tX]
            apply_type[OF B(2) tX] by auto
          also have ...=({⟨m,(r · (AAtm))⟩. m∈X}t)·_S(Btt) using apply_equality[OF
_ mr] tX by auto
          also have ...=∑[Tk;{{⟨m,(r · (AAtm))⟩. m∈X},Bt}] using sing
linComb_one_element[OF tX mr B(2)]
            by auto
          ultimately have r ·_S (∑[Tk;{AAt,Bt}])=∑[Tk;{{⟨m,(r · (AAtm))⟩.
m∈X},Bt}] by auto
        }
      }
      ultimately have  ∀AA∈X → R. ∀B∈X → ℳ. r ·_S (∑[Tk;{AA,B}]) =
∑[Tk;{{⟨x, r · (AA  x)⟩ . x ∈ X},B}]
        by auto
    }
    with tt have ∃t∈Tk. (∀AAt∈X → R.  ∀Bt∈X → ℳ.
                  r ·_S (∑[Tk - {t};{AAt,Bt}]) =
                  ∑[Tk - {t};{{⟨x, r · (AAt  x)⟩ . x ∈ X},Bt}]) ⟶
              (∀AAt∈X → R. ∀Bt∈X → ℳ.
                  r ·_S (∑[Tk;{AAt,Bt}]) = ∑[Tk;{{⟨x, r · (AAt  x)⟩
. x ∈ X},Bt}]) by auto
  }
  then have caseStep:∀A∈FinPow(X). A≠0 ⟶ (∃t∈A. (∀AAt∈X → R.
```

```
                    ∀Bt∈X → ℳ.
                       r ·S (∑[A - {t};{AAt,Bt}]) =
                       ∑[A - {t};{{⟨x, r · (AAt  x)⟩ . x ∈ X},Bt}]) ⟶
                 (∀AAt∈X → R.
                    ∀Bt∈X → ℳ.
                       r ·S (∑[A;{AAt,Bt}]) = ∑[A;{{⟨x, r · (AAt  x)⟩
. x ∈ X},Bt}])) by auto
  have ∀AAt∈X→R. ∀Bt∈X→ℳ. r ·S(∑[D;{AAt,Bt}]) =∑[D;{{⟨x,r·(AAtx)⟩.
x∈X},Bt}] using
    FinPow_ind_rem_one[where P=λD. (∀AAt∈X→R. ∀Bt∈X→ℳ. r ·S(∑[D;{AAt,Bt}])
=∑[D;{{⟨x,r·(AAtx)⟩. x∈X},Bt}]),
    OF case0 caseStep] assms(4) by auto
  with assms(1,2) show r ·S(∑[D;{AA,B}]) =∑[D;{{⟨x,r·(AAx)⟩. x∈X},B}]
by auto
qed
```

A linear combination can always be defined on a cardinal.

```
lemma(in module0) linComb_reorder_terms1:
  assumes AA:X→R B:X→ℳ D∈FinPow(X) g∈bij(|D|,D)
  shows (∑[D;{AA,B}])=∑[|D|;{AA O g,B O g}]
proof-
  from assms(3,4) have funf:g:|D|→X unfolding bij_def inj_def FinPow_def
using func1_1_L1B by auto
  have ABfun:AA O g:|D|→R B O g:|D|→ℳ using comp_fun[OF funf assms(1)]
comp_fun[OF funf assms(2)] by auto
  then have domAg:domain(AA O g)=|D| unfolding Pi_def by auto
  from assms(1) have domA:domain(AA)=X unfolding Pi_def by auto
  have comm:commsemigr(ℳ, A_M, domain(AA), {⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)})
    unfolding commsemigr_def using mod_ab_gr.abelian_group_axioms un-
folding IsAgroup_def
    IsAmonoid_def abelian_group_def group0_def abelian_group_axioms_def
    using coordinate_function[OF assms(1,2)] domA by auto
  {
    assume A:D=0
    then have D:∑[D;{AA,B}]=Θ using LinearComb_def assms(1-3) by auto
moreover
    from A assms(4) have |D|=0 unfolding bij_def inj_def Pi_def by auto
    moreover then have |D|∈FinPow(|D|) unfolding FinPow_def by auto
moreover
    note ABfun
    ultimately have thesis using LinearComb_def[of AA O g |D| B O g, of
|D|] by auto
  }
  moreover
  {
    assume A:D≠0
    then have eqD:∑[D;{AA,B}]=CommSetFold(A_M,{⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)},D)
using LinearComb_def[OF assms(1-3)] by auto
    have eqD1:CommSetFold(A_M,{⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)},D)=Fold1(A_M,{⟨m,(AAm)·S
```

```
(Bm)⟩. m∈domain(AA)} O g)
      using commsemigr.sum_over_set_bij[OF comm] assms(3) A
      domA assms(4) by blast
    {
      fix n assume n:n∈|D|
      then have T:({⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)} O g)n={⟨m,(AAm)·S (Bm)⟩.
m∈domain(AA)}( gn)
        using comp_fun_apply funf unfolding bij_def inj_def by auto
      have gn∈D using assms(4) unfolding bij_def inj_def using apply_type
n by auto
      then have gn∈domain(AA) using domA assms(3) unfolding FinPow_def
by auto
      then have {⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)}( gn)=(AA(gn))·S (B( gn))
using apply_equality[OF _ coordinate_function[OF assms(1,2)]]
        domA by auto
      also have ...=((AA O g)n)·S ((B O g)n) using comp_fun_apply funf
n by auto
      also have ...={⟨m,((AA O g)m)·S ((B O g)m)⟩. m∈|D|}n using apply_equality[OF
_ coordinate_function[OF comp_fun[OF funf assms(1)] comp_fun[OF funf assms(2)]]]
        n by auto
      ultimately have ({⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)} O g)n={⟨m,((AA
O g)m)·S ((B O g)m)⟩. m∈|D|}n using T by auto
    }
    then have ∀x∈|D|. ({⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)} O g)x={⟨m,((AA
O g)m)·S ((B O g)m)⟩. m∈|D|}x by auto
    then have eq:{⟨m,(AAm)·S (Bm)⟩. m∈domain(AA)} O g={⟨m,((AA O g)m)·S
((B O g)m)⟩. m∈|D|} using func_eq[OF comp_fun[OF funf coordinate_function[OF
assms(1) assms(2)]]
      coordinate_function[OF comp_fun[OF funf assms(1)] comp_fun[OF funf
assms(2)]]] domA by auto
    have R1:∑[D;{AA,B}]=Fold1(AM,{⟨m,((AA O g)m)·S ((B O g)m)⟩. m∈|D|})
      using trans[OF eqD eqD1] subst[OF eq, of λt. ∑[D;{AA,B}] = Fold1(AM,t)]
by blast
    from A have Dno:|D|≠0 using assms(4) unfolding bij_def surj_def by
auto
    have ||D||=|D| using cardinal_cong assms(4) unfolding eqpoll_def by
auto
    then have idf:id(|D|)∈bij(||D||,|D|) using id_bij by auto
    have comm:commsemigr(ℳ, AM, |D|, {⟨m, ((AA O g)  m) ·S ((B O g)
m)⟩ . m ∈ |D|})
      unfolding commsemigr_def using mod_ab_gr.abelian_group_axioms
      unfolding abelian_group_def group0_def IsAgroup_def IsAmonoid_def
      abelian_group_axioms_def using coordinate_function[OF comp_fun[OF
funf assms(1)] comp_fun[OF funf assms(2)]]
      by auto
    have sub:{⟨m, ((AA O g)  m) ·S ((B O g)  m)⟩ . m ∈ |D|} ⊆ |D|×ℳ
using
      coordinate_function[OF comp_fun[OF funf assms(1)] comp_fun[OF funf
assms(2)]]
```

```
      unfolding Pi_def by auto
    have finE:Finite(|D|) using assms(4,3) eqpoll_imp_Finite_iff unfold-
ing eqpoll_def FinPow_def by auto
    then have EFP:|D|∈FinPow(|D|) unfolding FinPow_def by auto
    then have eqE:∑[|D|;{AA O g,B O g}]=CommSetFold(A_M,{⟨m,((AA O g)m)·_S
((B O g)m)⟩. m∈domain(AA O g)},|D|) using LinearComb_def[OF comp_fun[OF
funf assms(1)]
      comp_fun[OF funf assms(2)]] Dno by auto
    also have ...=CommSetFold(A_M,{⟨m,((AA O g)m)·_S ((B O g)m)⟩. m∈|D|},|D|)
using domAg by auto
    also have ...=Fold1(A_M,{⟨m,((AA O g)m)·_S ((B O g)m)⟩. m∈|D|}) using
commsemigr.sum_over_set_bij[OF comm
      EFP Dno idf]
      subst[OF right_comp_id[of {⟨m,((AA O g)m)·_S ((B O g)m)⟩. m∈|D|} |D|
ℳ, OF sub],
        of λt. CommSetFold(A_M,{⟨m,((AA O g)m)·_S ((B O g)m)⟩. m∈|D|},|D|)
= Fold1(A_M,t)]
      by blast
    ultimately have ∑[|D|;{AA O g,B O g}]=Fold1(A_M,{⟨m,((AA O g)m)·_S
((B O g)m)⟩. m∈|D|})
      by (simp only:trans)
    with R1 have thesis by (simp only:trans sym)
  }
  ultimately show thesis by blast
qed
```

Actually a linear combination can be defined over any bijective set with the original set.

```
lemma(in module0) linComb_reorder_terms2:
  assumes AA:X→R B:X→ℳ D∈FinPow(X) g∈bij(E,D)
  shows (∑[D;{AA,B}])=∑[E;{AA O g,B O g}]
proof-
  from assms(3,4) have funf:g:E→X unfolding bij_def inj_def FinPow_def
using func1_1_L1B by auto
  have ABfun:AA O g:E→R B O g:E→ℳ using comp_fun[OF funf assms(1)]
comp_fun[OF funf assms(2)] by auto
  then have domAg:domain(AA O g)=E unfolding Pi_def by auto
  from assms(1) have domA:domain(AA)=X unfolding Pi_def by auto
  from assms(3-4) have finE:Finite(E) unfolding FinPow_def using eqpoll_imp_Finite_iff
unfolding eqpoll_def by auto
  then have |E|≈E using well_ord_cardinal_eqpoll Finite_imp_well_ord
by blast
  then obtain h where h:h∈bij(|E|,E) unfolding eqpoll_def by auto
  then have (g O h)∈bij(|E|,D) using comp_bij assms(4) by auto more-
over
  have ED:|E|=|D| using cardinal_cong assms(4) unfolding eqpoll_def by
auto
  ultimately have (g O h)∈bij(|D|,D) by auto
  then have (∑[D;{AA,B}])=(∑[|D|;{AA O (g O h),B O (g O h)}]) using
```

```
linComb_reorder_terms1 assms(1-3) by auto moreover
  from h have (∑ [E;{AA O g,B O g}])=(∑ [|E|;{(AA O g) O h,(B O g) O
h}]) using linComb_reorder_terms1 comp_fun[OF funf assms(1)] comp_fun[OF
funf assms(2)]
    finE unfolding FinPow_def by auto
  moreover note ED ultimately
  show thesis using comp_assoc by auto
qed
```

Restricting the defining functions to the domain set does nothing to the linear combination

```
corollary(in module0) linComb_restrict_coord:
  assumes AA:X→R B:X→ℳ D∈FinPow(X)
  shows (∑ [D;{AA,B}])=∑ [D;{restrict(AA,D),restrict(B,D)}]
  using linComb_reorder_terms2[OF assms id_bij] right_comp_id_any by auto
```

A linear combination can by defined with a natural number and functions with that number as domain.

```
corollary(in module0) linComb_nat:
  assumes AA:X→R B:X→ℳ D∈FinPow(X)
  shows ∃n∈nat. ∃A1∈n→R. ∃B1∈n→ℳ. ∑ [D;{AA,B}]=∑ [n;{A1,B1}] ∧ A1n=AAD
∧ B1n=BD
proof
  from assms(3) have fin:Finite(D) unfolding FinPow_def by auto
  then obtain n where n:n∈nat D≈n unfolding Finite_def by auto more-
over
  from fin have |D|≈D using well_ord_cardinal_eqpoll Finite_imp_well_ord
by blast
  ultimately have |D|≈n n∈nat using eqpoll_trans[of |D|] by auto
  then have n∈nat ||D||=|n| using cardinal_cong by auto
  then have D:|D|=n n∈nat using Card_cardinal_eq[OF Card_cardinal] Card_cardinal_eq[OF
nat_into_Card] by auto
  then show |D|∈nat by auto
  from n(2) D(1) obtain g where g:g∈bij(|D|,D) using eqpoll_sym un-
folding eqpoll_def by auto
  show ∃A1∈|D|→R. ∃B1∈|D|→ℳ. ∑ [D;{AA,B}]=∑ [|D|;{A1,B1}] ∧ A1|D|=AAD
∧ B1|D|=BD
  proof
    from g have gX:g:|D|→X unfolding bij_def inj_def using assms(3)
unfolding FinPow_def using func1_1_L1B by auto
    then show (AA O g):|D|→R using comp_fun assms(1) by auto
    show ∃B1∈|D|→ℳ. ∑ [D;{AA,B}]=∑ [|D|;{AA O g,B1}] ∧ (AA O g)|D|=AAD
∧ B1|D|=BD
    proof
      show (B O g):|D|→ℳ using comp_fun assms(2) gX by auto
      have ∑ [D;{AA,B}]=∑ [|D|;{AA O g,B O g}] using g linComb_reorder_terms1
assms func1_1_L1B by auto
      moreover have (AA O g)|D|=AA(g|D|) using image_comp by auto
```

```
      then have (AA O g)|D|=AAD using g unfolding bij_def using surj_range_image_domain
by auto
      moreover have (B O g)|D|=B(g|D|) using image_comp by auto
      then have (B O g)|D|=BD using g unfolding bij_def using surj_range_image_domain
by auto
      ultimately show ∑[D;{AA,B}]=∑[|D|;{AA O g,B O g}] ∧ (AA O g)|D|=AAD
∧ (B O g)|D|=BD by auto
    qed
  qed
qed
```

52.3.1 Adding linear combinations

Adding a linear combination defined over ∅ leaves it as is

```
lemma(in module0) linComb_sum_base_induct1:
  assumes AA:X→R B:X→ℳ D∈FinPow(X) AA1:Y→R B1:Y→ℳ
  shows (∑[D;{AA,B}])+V(∑[0;{AA1,B1}])=∑[D;{AA,B}]
proof-
  from assms(1-3) have ∑[D;{AA,B}]∈ℳ using linComb_is_in_module by
auto
  then have eq:(∑[D;{AA,B}])+VΘ=∑[D;{AA,B}] using zero_neutral(1)
    by auto moreover
  have ff:0∈FinPow(Y) unfolding FinPow_def by auto
  from eq show thesis using LinearComb_def[OF assms(4-5) ff] by auto
qed
```

Applying a product of 1× to the defining set computes the same linear combination; since they are bijective sets

```
lemma(in module0) linComb_sum_base_induct2:
  assumes AA:X→R B:X→ℳ D∈FinPow(X)
  shows (∑[D;{AA,B}])=∑[{0}×D;{{⟨⟨0,x⟩,AAx⟩. x∈X},{⟨⟨0,x⟩,Bx⟩. x∈X}}]
and
  (∑[D;{AA,B}])=∑[{0}×D;{restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D),restrict({⟨⟨0,x⟩,Bx⟩.
x∈X},{0}×D)}]
proof-
  let g={⟨⟨0,d⟩,d⟩. d∈D}
  from assms(3) have sub:D⊆X unfolding FinPow_def by auto
  have gfun:g:{0}×D→D unfolding Pi_def function_def by blast
  then have gfunX:g:{0}×D→X using sub func1_1_L1B by auto
  from gfun have g∈surj({0}×D,D) unfolding surj_def using apply_equality[OF
_ gfun] by blast moreover
  {
    fix w y assume gw=gy w∈{0}×D y∈{0}×D
    then obtain dw dy where w=⟨0,dw⟩ y=⟨0,dy⟩ gw=gy dw∈D dy∈D by auto
    then have dw=dy w=⟨0,dw⟩ y=⟨0,dy⟩ using apply_equality[OF _ gfun,
of w dw] apply_equality[OF _ gfun, of y dy] by auto
    then have w=y by auto
  }
```

then have g∈inj({0}×D,D) unfolding inj_def using gfun by auto ultimately
have gbij:g∈bij({0}×D,D) unfolding bij_def by auto
with assms have A1:(∑ [D;{AA,B}])=∑ [{0}×D;{AA O g,B O g}] using linComb_reorder_terms2 by auto
from gbij have Finite({0}×D) using assms(3) eqpoll_imp_Finite_iff unfolding FinPow_def eqpoll_def by auto
then have fin:{0}×D∈FinPow({0}×X) unfolding FinPow_def using sub by auto
have A0:{⟨⟨0,x⟩,AAx⟩. x∈X}:{0}×X→R unfolding Pi_def function_def using apply_type[OF assms(1)] by auto
have B0:{⟨⟨0,x⟩,Bx⟩. x∈X}:{0}×X→$\mathcal{M}$ unfolding Pi_def function_def using apply_type[OF assms(2)] by auto
{
fix r assume r∈{0}×D
then obtain rd where rd:r=⟨0,rd⟩ rd∈D by auto
then have (AA O g)r=AArd using comp_fun_apply gfun apply_equality by auto
also have ...={⟨⟨0,x⟩,AAx⟩. x∈X}⟨0,rd⟩ using apply_equality[OF _ A0] sub rd(2) by auto
also have ...=restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)⟨0,rd⟩ using restrict rd(2) by auto
ultimately have (AA O g)r=restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)r using rd(1) by auto
}
then have ∀r∈{0}×D. (AA O g)r=restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)r by auto moreover
have AA O g:{0}×D→R using gfunX assms(1) comp_fun by auto moreover
have {0}×D⊆{0}×X using sub by auto
then have restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D):{0}×D→R using A0 restrict_fun by auto ultimately
have f1:(AA O g)=restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D) using func_eq[of AA O g {0}×D R restrict({⟨⟨0, x⟩, AA x⟩ . x ∈ X}, {0} × D)]
by auto
{
fix r assume r∈{0}×D
then obtain rd where rd:r=⟨0,rd⟩ rd∈D by auto
then have (B O g)r=Brd using comp_fun_apply gfun apply_equality by auto
also have ...={⟨⟨0,x⟩,Bx⟩. x∈X}⟨0,rd⟩ using apply_equality[OF _ B0] sub rd(2) by auto
also have ...=restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)⟨0,rd⟩ using restrict rd(2) by auto
ultimately have (B O g)r=restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)r using rd(1) by auto
}
then have ∀r∈{0}×D. (B O g)r=restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)r by auto moreover
have B O g:{0}×D→$\mathcal{M}$ using gfunX assms(2) comp_fun by auto moreover

```
  have {0}×D⊆{0}×X using sub by auto
  then have restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D):{0}×D→ℳ using B0 restrict_fun
by auto ultimately
  have (B 0 g)=restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D) using func_eq[of B 0
g {0}×D ℳ restrict({⟨⟨0, x⟩, B  x⟩ . x ∈ X}, {0} × D)]
    by auto
  with A1 f1 show (∑[D;{AA,B}])=∑[{0}×D;{restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D),restrict({⟨⟨
x∈X},{0}×D)}] by auto
  also have ...=∑[{0}×D;{{⟨⟨0,x⟩,AAx⟩. x∈X},{⟨⟨0,x⟩,Bx⟩. x∈X}}] using linComb_restrict_coord
A0 B0 fin] by auto
  ultimately show (∑[D;{AA,B}])=∑[{0}×D;{{⟨⟨0,x⟩,AAx⟩. x∈X},{⟨⟨0,x⟩,Bx⟩.
x∈X}}] by auto
qed
```

Then, we can model adding a liner combination on the empty set as a linear combination of the disjoint union of sets

```
lemma(in module0) linComb_sum_base_induct:
  assumes AA:X→R B:X→ℳ D∈FinPow(X) AA1:Y→R B1:Y→ℳ
  shows (∑[D;{AA,B}])+_V(∑[0;{AA1,B1}])=∑[D+0;{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩.
x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}]
proof-
  from assms(3) have sub:D⊆X unfolding FinPow_def by auto
  then have sub2:{0}×D⊆{0}×X by auto
  then have sub3:{0}×D∈Pow(X+Y) unfolding sum_def by auto
  let g={⟨⟨0,x⟩,x⟩. x∈D}
  have gfun:g:{0}×D→D unfolding Pi_def function_def by blast
  then have g∈surj({0}×D,D) unfolding surj_def using apply_equality by
auto moreover
  {
    fix w y assume gw=gy w∈{0}×D y∈{0}×D
    then obtain dw dy where w=⟨0,dw⟩ y=⟨0,dy⟩ gw=gy dw∈D dy∈D by auto
    then have dw=dy w=⟨0,dw⟩ y=⟨0,dy⟩ using apply_equality[OF _ gfun,
of w dw] apply_equality[OF _ gfun, of y dy] by auto
    then have w=y by auto
  }
  then have g∈inj({0}×D,D) unfolding inj_def using gfun by auto ulti-
mately
  have gbij:g∈bij({0}×D,D) unfolding bij_def by auto
  then have Finite({0}×D) using assms(3) eqpoll_imp_Finite_iff unfold-
ing FinPow_def eqpoll_def by auto
  with sub3 have finpow0D:{0}×D∈FinPow(X+Y) unfolding FinPow_def by
auto
  have A0:{⟨⟨0,x⟩,AAx⟩. x∈X}:{0}×X→R unfolding Pi_def function_def us-
ing apply_type[OF assms(1)] by auto
  have A1:{⟨⟨1,x⟩,AA1x⟩. x∈Y}:{1}×Y→R unfolding Pi_def function_def us-
ing apply_type[OF assms(4)] by auto
  have domA0:domain({⟨⟨0,x⟩,AAx⟩. x∈X})={0}×X by auto
  have domA1:domain({⟨⟨1,x⟩,AA1x⟩. x∈Y})={1}×Y by auto
  have A:{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y}:X+Y→R using fun_disjoint_Un[OF
```

A0 A1] **unfolding** sum_def **by** auto

have B0:{⟨⟨0,x⟩,Bx⟩. x∈X}:{0}×X→$\mathcal{M}$ **unfolding** Pi_def function_def **using** apply_type[OF assms(2)] **by** auto

have B1:{⟨⟨1,x⟩,B1x⟩. x∈Y}:{1}×Y→$\mathcal{M}$ **unfolding** Pi_def function_def **using** apply_type[OF assms(5)] **by** auto

then have domB0:domain({⟨⟨0,x⟩,Bx⟩. x∈X})={0}×X **unfolding** Pi_def **by** auto

then have domB1:domain({⟨⟨1,x⟩,B1x⟩. x∈Y})={1}×Y **unfolding** Pi_def **by** auto

have B:{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}:X+Y→$\mathcal{M}$ **using** fun_disjoint_Un[OF B0 B1] **unfolding** sum_def **by** auto

have (∑[D;{AA,B}])$+_V$(∑[0;{AA1,B1}])=∑[D;{AA,B}] **using** linComb_sum_base_induct1 assms **by** auto

also have ...=∑[{0}×D;{restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D),restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)}] **using** linComb_sum_base_induct2(2) assms(1-3) **by** auto

ultimately have eq1:(∑[D;{AA,B}])$+_V$(∑[0;{AA1,B1}])=∑[{0}×D;{restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D),restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)}] **by** auto

{

fix s **assume** s∈{0}×D

then obtain ds **where** ds:ds∈D s=⟨0,ds⟩ **by** auto

then have restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)s={⟨⟨0,x⟩,AAx⟩. x∈X}s **using** restrict **by** auto

also have ...=({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s **using** fun_disjoint_apply1 domA1 ds(2) **by** auto

also have ...=restrict({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{0}×D)s **using** restrict ds **by** auto

ultimately have restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)s=restrict({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{0}×D)s **by** auto

}

then have tot:∀s∈{0}×D. restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)s=restrict({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{0}×D)s **by** auto **moreover**

have f1:restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D):{0}×D→R **using** restrict_fun A0 sub2 **by** auto **moreover**

have f2:restrict({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{0}×D):{0}×D→R **using** restrict_fun[OF fun_disjoint_Un[OF A0 A1]] sub2 **by** auto

ultimately have fA:restrict({⟨⟨0,x⟩,AAx⟩. x∈X},{0}×D)=restrict({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{0}×D) **using** func_eq[OF f1 f2] **by** auto

{

fix s **assume** s∈{0}×D

then obtain ds **where** ds:ds∈D s=⟨0,ds⟩ **by** auto

then have restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)s={⟨⟨0,x⟩,Bx⟩. x∈X}s **using** restrict **by** auto

also have ...=({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s **using** fun_disjoint_apply1 domB1 ds(2) **by** auto

also have ...=restrict({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y},{0}×D)s **using** restrict ds **by** auto

ultimately have restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)s=restrict({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y},{0}×D)s **by** auto

}

```
  then have tot:∀s∈{0}×D. restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)s=restrict({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y},{0}×D)s by auto moreover
  have f1:restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D):{0}×D→ℳ using restrict_fun
B0 sub2 by auto moreover
  have f2:restrict({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y},{0}×D):{0}×D→ℳ
using restrict_fun[OF fun_disjoint_Un[OF B0 B1]] sub2 by auto
  ultimately have fB:restrict({⟨⟨0,x⟩,Bx⟩. x∈X},{0}×D)=restrict({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y},{0}×D) using func_eq[OF f1 f2] by auto
  with fA eq1 have (∑[D;{AA,B}])+_V(∑[0;{AA1,B1}])=∑[{0}×D;{restrict({⟨⟨0,x⟩,AAx⟩.
x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{0}×D),restrict({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩.
x∈Y},{0}×D)}]
    by auto
  also have ...=∑[{0}×D;{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}] using linComb_restrict_coord[OF A B finpow0D]
    by auto
  also have ...=∑[D+0;{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}] unfolding sum_def by auto
  ultimately show thesis by auto
qed
```

An element of the set for the linear combination can be removed and add it using group addition.

```
lemma(in module0) sum_one_element:
  assumes AA:X→R B:X→ℳ D∈FinPow(X) t∈D
  shows (∑[D;{AA,B}])=(∑[D-{t};{AA,B}])+_V({⟨k,(AAk)·_S(Bk)⟩. k∈X}t)
proof-
  from assms(1,2) have af:{⟨k,(AAk)·_S(Bk)⟩. k∈X}:X→ℳ using coordinate_function
by auto
  from assms(3) have sub:D⊆X unfolding FinPow_def by auto
  then have tX:t∈X using assms(4) by auto
  have dom:domain(AA)=X using assms(1) Pi_def by auto
  {
    assume A:D-{t}=0
    with assms(4) have D={t} by auto
    then have (∑[D;{AA,B}])=∑[{t};{AA,B}] by auto
    also have ...=(AAt)·_S(Bt) using linComb_one_element sub assms(1,2,4)
by auto
    also have ...={⟨k,(AAk)·_S(Bk)⟩. k∈X}t using af assms(4) sub apply_equality
by auto
    also have ...=Θ +_V({⟨k,(AAk)·_S(Bk)⟩. k∈X}t) using zero_neutral(2)
      apply_type[OF af] assms(4) sub by auto
    also have ...=(∑[D-{t};{AA,B}])+_V({⟨k,(AAk)·_S(Bk)⟩. k∈X}t) using A
LinearComb_def[OF assms(1,2), of D-{t}]
      unfolding FinPow_def by auto
    ultimately have thesis by auto
  }
  moreover
  {
    assume A:D-{t}≠0
```

```
    then have D:D≠0 by auto
    have comm:commsemigr(𝓜, A_M, X, {⟨k,(AAk)·_S(Bk)⟩. k∈X})
      unfolding commsemigr_def using mod_ab_gr.abelian_group_axioms
      unfolding abelian_group_def abelian_group_axioms_def group0_def
      IsAgroup_def IsAmonoid_def using af by auto
    have fin:D-{t}∈FinPow(X) using assms(3) unfolding FinPow_def using
subset_Finite[of D-{t} D] by auto
    have (D-{t})∪{t}=D X-(D-{t})=(X-D)∪{t} using assms(4) sub by auto
    then have CommSetFold(A_M,{⟨k,(AAk)·_S(Bk)⟩. k∈X},D)=CommSetFold(A_M,{⟨k,(AAk)·_S(Bk)⟩.
k∈X},D-{t})
      +_V({⟨k,(AAk)·_S(Bk)⟩. k∈X}t) using commsemigr.sum_over_set_add_point(2)[OF
comm
      fin A] by auto
    also have ...=(∑[D-{t};{AA,B}])+_V({⟨k,(AAk)·_S(Bk)⟩. k∈X}t) using LinearComb_def[OF
assms(1,2) fin] A
      dom by auto
    ultimately have CommSetFold(A_M,{⟨k,(AAk)·_S(Bk)⟩. k∈X},D)=(∑[D-{t};{AA,B}])+_V({⟨k,(AAk)
k∈X}t) by auto moreover
    then have thesis unfolding LinearComb_def[OF assms(1-3)] using D dom
by auto
  }
  ultimately show thesis by blast
qed
```

A small technical lemma to proof by induction on finite sets that the addition of linear combinations is a linear combination

```
lemma(in module0) linComb_sum_ind_step:
  assumes AA:X→R B:X→𝓜 D∈FinPow(X) E∈FinPow(Y) AA1:Y→R B1:Y→𝓜 t∈E
D≠0
    (∑[D;{AA,B}])+_V(∑[E-{t};{AA1,B1}])=∑[D+(E-{t});{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩.
x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}]
  shows (∑[D;{AA,B}])+_V(∑[E;{AA1,B1}])=∑[D+E;{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩.
x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}]
proof-
  have a1f:{⟨k,(AA1k)·_S(B1k)⟩. k∈Y}:Y→𝓜 using coordinate_function assms(5,6)
by auto
  with assms(4,7) have p1:({⟨k,(AA1k)·_S(B1k)⟩. k∈Y}t)∈𝓜 unfolding FinPow_def
using apply_type by auto
  have p2:∑[D;{AA,B}]∈𝓜 using linComb_is_in_module assms(1-3) by auto
  have E-{t}∈FinPow(Y) using assms(4,7) unfolding FinPow_def using subset_Finite[of
E-{t} E] by auto
  then have p3:∑[E-{t};{AA1,B1}]∈𝓜 using linComb_is_in_module assms(5,6)
by auto
  have ∑[E;{AA1,B1}]=(∑[E-{t};{AA1,B1}])+_V({⟨k,(AA1k)·_S(B1k)⟩. k∈Y}t)
using sum_one_element[OF assms(5,6,4,7)].
  then have (∑[D;{AA,B}])+_V(∑[E;{AA1,B1}])=(∑[D;{AA,B}])+_V((∑[E-{t};{AA1,B1}])+_V({⟨k,
k∈Y}t)) by auto
  also have ...=((∑[D;{AA,B}])+_V(∑[E-{t};{AA1,B1}]))+_V({⟨k,(AA1k)·_S(B1k)⟩.
k∈Y}t)
```

using p1 p2 p3 mod_ab_gr.group_oper_assoc **by** auto
also have ...=(∑ [D+(E-{t});{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}])+$_V$({⟨k,(AA1k)·$_S$(B1k)⟩. k∈Y}t)
using assms(9) **by** auto
ultimately have (∑ [D;{AA,B}])+$_V$(∑ [E;{AA1,B1}])=(∑ [D+(E-{t});{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}])+$_V$({⟨k,(AA1k)·$_S$(B1k)⟩. k∈Y}t) **by** auto
moreover have {⟨k,(AA1k)·$_S$(B1k)⟩. k∈Y}t=(AA1t)·$_S$(B1t) **using** apply_equality[OF _ a1f] assms(7,4) **unfolding** FinPow_def **by** auto **moreover**
{
have dA:domain({⟨⟨0,x⟩,AAx⟩. x∈X})={⟨0,x⟩. x∈X} **unfolding** domain_def **by** auto
have dA1:domain({⟨⟨1,x⟩,AA1x⟩. x∈Y})={⟨1,x⟩. x∈Y} **unfolding** domain_def **by** auto
have ⟨1,t⟩∉domain({⟨⟨0,x⟩,AAx⟩. x∈X}) **by** auto
then have ({⟨⟨1,x⟩,AA1x⟩. x∈Y}∪{⟨⟨0,x⟩,AAx⟩. x∈X})⟨1,t⟩={⟨⟨1,x⟩,AA1x⟩. x∈Y}⟨1,t⟩ **using** fun_disjoint_apply1[of ⟨1,t⟩ {⟨⟨0,x⟩,AAx⟩. x∈X}] **by** auto
moreover have {⟨⟨1,x⟩,AA1x⟩. x∈Y}∪{⟨⟨0,x⟩,AAx⟩. x∈X}={⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y} **by** auto
ultimately have ({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,t⟩={⟨⟨1,x⟩,AA1x⟩. x∈Y}⟨1,t⟩ **by** auto **moreover**
have {⟨⟨1,x⟩,AA1x⟩. x∈Y}:{1}×Y→R **unfolding** Pi_def function_def **using** apply_type[OF assms(5)] **by** auto
then have {⟨⟨1,x⟩,AA1x⟩. x∈Y}⟨1,t⟩=(AA1t) **using** apply_equality assms(7,4) **unfolding** FinPow_def **by** auto
ultimately have e1:({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,t⟩=(AA1t) **by** auto
have dB:domain({⟨⟨0,x⟩,Bx⟩. x∈X})={⟨0,x⟩. x∈X} **unfolding** domain_def **by** auto
have dB1:domain({⟨⟨1,x⟩,B1x⟩. x∈Y})={⟨1,x⟩. x∈Y} **unfolding** domain_def **by** auto
have ⟨1,t⟩∉domain({⟨⟨0,x⟩,Bx⟩. x∈X}) **by** auto
then have ({⟨⟨1,x⟩,B1x⟩. x∈Y}∪{⟨⟨0,x⟩,Bx⟩. x∈X})⟨1,t⟩={⟨⟨1,x⟩,B1x⟩. x∈Y}⟨1,t⟩ **using** fun_disjoint_apply1[of ⟨1,t⟩ {⟨⟨0,x⟩,Bx⟩. x∈X}] **by** auto
moreover have {⟨⟨1,x⟩,B1x⟩. x∈Y}∪{⟨⟨0,x⟩,Bx⟩. x∈X}={⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y} **by** auto
ultimately have ({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,t⟩={⟨⟨1,x⟩,B1x⟩. x∈Y}⟨1,t⟩ **by** auto **moreover**
have {⟨⟨1,x⟩,B1x⟩. x∈Y}:{1}×Y→$\mathcal{M}$ **unfolding** Pi_def function_def **using** apply_type[OF assms(6)] **by** auto
then have {⟨⟨1,x⟩,B1x⟩. x∈Y}⟨1,t⟩=(B1t) **using** apply_equality assms(7,4) **unfolding** FinPow_def **by** auto
ultimately have e2:({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,t⟩=(B1t) **by** auto
with e1 **have** (({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,t⟩)·$_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,t⟩)=(AA1t)·$_S$(B1t) **by** auto
moreover have tXY:⟨1,t⟩∈X+Y **unfolding** sum_def **using** assms(7,4) **unfolding** FinPow_def **by** auto
{

```
      fix s w assume as:⟨s,w⟩∈{⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y}
      then have s:s∈X+Y and w:w=(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s) by auto
      then have ss:s∈{0}×X ∪ {1}×Y unfolding sum_def by auto
      {
        assume s∈{0}×X
        then obtain r where r:r∈X s=⟨0,r⟩ by auto
        with s have a:⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨0,r⟩)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨0,r⟩)⟩∈
          {⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} by auto
        have ({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨0,r⟩={⟨⟨0,x⟩,AAx⟩.
x∈X}⟨0,r⟩ using fun_disjoint_apply1[of ⟨0,r⟩
          {⟨⟨1,x⟩,AA1x⟩. x∈Y}] by auto
        also have ...=AAr using r(1) apply_equality[of ⟨0,r⟩ AAr {⟨⟨0,x⟩,AAx⟩.
x∈X} {0}×X λp. R] unfolding Pi_def function_def
          using apply_type[OF assms(1)] by auto
        ultimately have AAr:({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨0,r⟩=AAr
by auto
        with a have a1:⟨s,(AAr)·S(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨0,r⟩)⟩∈
          {⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} by auto
        have ({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨0,r⟩={⟨⟨0,x⟩,Bx⟩. x∈X}⟨0,r⟩
using fun_disjoint_apply1[of ⟨0,r⟩
          {⟨⟨1,x⟩,B1x⟩. x∈Y}] by auto
        also have ...=Br using r(1) apply_equality[of ⟨0,r⟩ Br {⟨⟨0,x⟩,Bx⟩.
x∈X} {0}×X λp. M] unfolding Pi_def function_def
          using apply_type[OF assms(2)] by auto
        ultimately have Br:({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨0,r⟩=Br
by auto
        with a1 have ⟨s,(AAr)·S(Br)⟩∈
          {⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} by auto moreover
        have (AAr)·S(Br)∈M using apply_type[OF coordinate_function[OF
assms(1,2)] r(1)] apply_equality[OF _ coordinate_function[OF assms(1,2)]]
r(1) by auto
        ultimately have ⟨s,(AAr)·S(Br)⟩∈(X+Y)×M using s by auto more-
over
        from w r(2) AAr Br have w=(AAr)·S(Br) by auto
        ultimately have ⟨s,w⟩∈(X+Y)×M by auto
      }
      moreover
      {
        assume s∉{0}×X
        with ss have s∈{1}×Y by auto
        then obtain r where r:r∈Y s=⟨1,r⟩ by auto
        with s have a:⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,r⟩)·S(({⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,r⟩)⟩∈
```

{⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} **by** auto
have ({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,r⟩={⟨⟨1,x⟩,AA1x⟩. x∈Y}⟨1,r⟩ **using** fun_disjoint_apply2[of ⟨1,r⟩
{⟨⟨0,x⟩,AAx⟩. x∈X}] **by** auto
also have ...=AA1r **using** r(1) apply_equality[of ⟨1,r⟩ AA1r {⟨⟨1,x⟩,AA1x⟩. x∈Y} {1}×Y λp. R] **unfolding** Pi_def function_def
using apply_type[OF assms(5)] **by** auto
ultimately have AAr:({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,r⟩=AA1r **by** auto
with a **have** a1:⟨s,(AA1r)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,r⟩)⟩∈
{⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} **by** auto
have ({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,r⟩={⟨⟨1,x⟩,B1x⟩. x∈Y}⟨1,r⟩ **using** fun_disjoint_apply2[of ⟨1,r⟩
{⟨⟨0,x⟩,Bx⟩. x∈X}] **by** auto
also have ...=B1r **using** r(1) apply_equality[of ⟨1,r⟩ B1r {⟨⟨1,x⟩,B1x⟩. x∈Y} {1}×Y λp. $\mathcal{M}$] **unfolding** Pi_def function_def
using apply_type[OF assms(6)] **by** auto
ultimately have Br:({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,r⟩=B1r **by** auto
with a1 **have** ⟨s,(AA1r)$\cdot_S$(B1r)⟩∈
{⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} **by** auto **moreover**
have (AA1r)$\cdot_S$(B1r)∈$\mathcal{M}$ **using** apply_type[OF coordinate_function[OF assms(5,6)] r(1)] apply_equality[OF _ coordinate_function[OF assms(5,6)]] r(1) **by** auto
ultimately have ⟨s,(AA1r)$\cdot_S$(B1r)⟩∈(X+Y)×$\mathcal{M}$ **using** s **by** auto **moreover**
from w r(2) AAr Br **have** w=(AA1r)$\cdot_S$(B1r) **by** auto
ultimately have ⟨s,w⟩∈(X+Y)×$\mathcal{M}$ **by** auto
}
ultimately have ⟨s,w⟩∈(X+Y)×$\mathcal{M}$ **by** auto
}
then have {⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y}⊆(X+Y)×$\mathcal{M}$ **by** auto
then have fun:{⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y}:X+Y→$\mathcal{M}$
unfolding Pi_def function_def **by** auto
from tXY **have** pp:⟨⟨1,t⟩,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,t⟩)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,t⟩)⟩∈
{⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y} **by** auto
have {⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y}⟨1,t⟩=
(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})⟨1,t⟩)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})⟨1,t⟩) **using** apply_equality[OF pp fun] **by** auto
ultimately have {⟨s,(({⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y})s)$\cdot_S$(({⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y})s)⟩. s∈X+Y}⟨1,t⟩=(AA1t)$\cdot_S$(B1t)

```
      by auto
  }
  ultimately have (∑[D;{AA,B}]) +V (∑[E;{AA1,B1}]) =
    (∑[D+(E-{t});{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩.
x∈Y}}]) +V
    (({⟨s, ((({⟨⟨0, x⟩, AA  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y})  s)
·S (({⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩ . x ∈ Y})  s))⟩ .
     s ∈ X + Y})  ⟨1, t⟩) by auto moreover
  have D+(E-{t})=(D+E)-{⟨1,t⟩} unfolding sum_def by auto ultimately
  have A1:(∑[D;{AA,B}]) +V (∑[E;{AA1,B1}]) =
    (∑[(D+E)-{⟨1,t⟩};{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩.
x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}]) +V
    (({⟨s, ((({⟨⟨0, x⟩, AA  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y})  s)
·S (({⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩ . x ∈ Y})  s))⟩ .
     s ∈ X + Y})  ⟨1, t⟩) by auto
  have f1:{⟨⟨0, x⟩, AA  x⟩ . x ∈ X}:{0}×X→R using apply_type[OF assms(1)]
unfolding Pi_def function_def by auto
  have f2:{⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y}:{1}×Y→R using apply_type[OF assms(5)]
unfolding Pi_def function_def by auto
  have ({0}×X)∩({1}×Y)=0 by auto
  then have ffA:({⟨⟨0, x⟩, AA  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y}):X+Y→R
unfolding sum_def using fun_disjoint_Un[OF f1 f2] by auto
  have f1:{⟨⟨0, x⟩, B  x⟩ . x ∈ X}:{0}×X→M using apply_type[OF assms(2)]
unfolding Pi_def function_def by auto
  have f2:{⟨⟨1, x⟩, B1  x⟩ . x ∈ Y}:{1}×Y→M using apply_type[OF assms(6)]
unfolding Pi_def function_def by auto
  have ({0}×X)∩({1}×Y)=0 by auto
  then have ffB:({⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩ . x ∈ Y}):X+Y→M
unfolding sum_def using fun_disjoint_Un[OF f1 f2] by auto
  have D+E⊆X+Y using assms(3,4) unfolding FinPow_def sum_def by auto
moreover
  obtain nd ne where nd∈nat D≈nd ne∈nat E≈ne using assms(3,4) unfold-
ing FinPow_def Finite_def by auto
  then have D+E≈nd+ne nd∈nat ne∈nat using sum_eqpoll_cong by auto
  then have D+E≈nd #+ ne using nat_sum_eqpoll_sum eqpoll_trans by auto
  then have ∃n∈nat. D+E≈n using add_type by auto
  then have Finite(D+E) unfolding Finite_def by auto
  ultimately have fin:D+E∈FinPow(X+Y) unfolding FinPow_def by auto
  from assms(7) have ⟨1,t⟩∈D+E unfolding sum_def by auto
  with A1 show thesis using sum_one_element[OF ffA ffB fin] by auto
qed
```

The addition of two linear combinations is a linear combination

```
theorem(in module0) linComb_sum:
  assumes AA:X→R AA1:Y→R B:X→M B1:Y→M D≠0 D∈FinPow(X) E∈FinPow(Y)
  shows (∑[D;{AA,B}])+V(∑[E;{AA1,B1}])=∑[D+E;{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩.
x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}]
proof-
  {
```

```
    fix 𝔄 𝔅 𝔄1 𝔅1 assume fun:𝔄:X→R 𝔅:X→ℳ 𝔄1:Y→R 𝔅1:Y→ℳ
    have ((∑[D;{𝔄,𝔅}]) +V(∑[0;{𝔄1,𝔅1}])=∑[D+0;{{⟨⟨0,x⟩,𝔄x⟩. x∈X}∪{⟨⟨1,x⟩,𝔄1x⟩.
x∈Y},{⟨⟨0,x⟩,𝔅x⟩. x∈X}∪{⟨⟨1,x⟩,𝔅1x⟩. x∈Y}}]) using linComb_sum_base_induct[OF
fun(1,2) assms(6) fun(3,4)]
      by auto
  }
  then have base:∀AA∈X → R.
        ∀B∈X → ℳ.
           ∀AA1∈Y → R.
              ∀B1∈Y → ℳ.
                 (∑[D;{AA,B}]) +V (∑[0;{AA1,B1}]) =
                 ∑[D + 0;{{⟨⟨0, x⟩, AA  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, AA1  x⟩
. x ∈ Y},{⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩ . x ∈ Y}}] by auto
  {
    fix ℜ assume R:ℜ∈FinPow(Y) ℜ≠0
    then obtain r where r:r∈ℜ by auto
    {
      fix 𝔄 𝔅 𝔄1 𝔅1 assume fun:𝔄:X→R 𝔅:X→ℳ 𝔄1:Y→R 𝔅1:Y→ℳ and

        step:∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. (∑[D;{AA,B}])
+V (∑[ℜ - {r};{AA1,B1}])=(∑[D+(ℜ - {r});{{⟨⟨0, x⟩, AA  x⟩ . x ∈ X}
∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y},{⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩
. x ∈ Y}}])
      then have (∑[D;{𝔄,𝔅}]) +V (∑[ℜ;{𝔄1,𝔅1}])=(∑[D+ℜ;{{⟨⟨0, x⟩,
𝔄  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, 𝔄1  x⟩ . x ∈ Y},{⟨⟨0, x⟩, 𝔅  x⟩ . x ∈ X} ∪
{⟨⟨1, x⟩, 𝔅1  x⟩ . x ∈ Y}}]) using linComb_sum_ind_step[OF fun(1,2) assms(6)
R(1) fun(3,4) r assms(5)]
        by auto
    }
    then have (∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. (∑[D;{AA,B}])
+V (∑[ℜ - {r};{AA1,B1}])=(∑[D+(ℜ - {r});{{⟨⟨0, x⟩, AA  x⟩ . x ∈ X}
∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y},{⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩
. x ∈ Y}}])) ⟶
      (∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. ((∑[D;{AA,B}]) +V(∑[ℜ;{AA1,B1}])=∑[
x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}])) by auto
    then have ∃r∈ℜ. (∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. (∑[D;{AA,B}])
+V (∑[ℜ - {r};{AA1,B1}])=(∑[D+(ℜ - {r});{{⟨⟨0, x⟩, AA  x⟩ . x ∈ X}
∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y},{⟨⟨0, x⟩, B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩
. x ∈ Y}}])) ⟶
      (∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. ((∑[D;{AA,B}]) +V(∑[ℜ;{AA1,B1}])=∑[
x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}])) using
r by auto
  }
  then have indstep:∀ℜ∈FinPow(Y). ℜ≠0 ⟶ (∃r∈ℜ. (∀AA∈X→R. ∀B∈X→ℳ.
∀AA1∈Y→R. ∀B1∈Y→ℳ. (∑[D;{AA,B}]) +V (∑[ℜ - {r};{AA1,B1}])=(∑[D+(ℜ
- {r});{{⟨⟨0, x⟩, AA  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, AA1  x⟩ . x ∈ Y},{⟨⟨0, x⟩,
B  x⟩ . x ∈ X} ∪ {⟨⟨1, x⟩, B1  x⟩ . x ∈ Y}}])) ⟶
    (∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. ((∑[D;{AA,B}]) +V(∑[ℜ;{AA1,B1}])=∑[D+
x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}]))) by auto
```

```
  have ∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R. ∀B1∈Y→ℳ. ((∑[D;{AA,B}]) +V (∑[E;{AA1,B1}])=∑[
x∈X}∪{⟨⟨1,x⟩,AA1x⟩. x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}])
    using FinPow_ind_rem_one[where P=λE. (∀AA∈X→R. ∀B∈X→ℳ. ∀AA1∈Y→R.
∀B1∈Y→ℳ. ((∑[D;{AA,B}]) +V (∑[E;{AA1,B1}])=∑[D+E;{{⟨⟨0,x⟩,AAx⟩. x∈X}∪{⟨⟨1,x⟩,AA1x⟩.
x∈Y},{⟨⟨0,x⟩,Bx⟩. x∈X}∪{⟨⟨1,x⟩,B1x⟩. x∈Y}}])),
    OF base indstep assms(7)].
  with assms(1-4) show thesis by auto
qed
```

52.3.2 Linear dependency

Now, we have the conditions to define what linear independence means:

```
definition(in module0)
  LinInde (_{is linearly independent} 89)
  where 𝒯⊆ℳ ⟹ 𝒯{is linearly independent} ≡ (∀X∈nat. ∀AA∈X→R. ∀B∈inj(X,𝒯).
((∑[X;{AA,B}] = Θ ) ) ⟶ (∀m∈X. AAm=0))
```

If a set has the zero element, then it is not linearly independent.

```
theorem(in module0) zero_set_dependent:
  assumes Θ∈T T⊆ℳ R≠{0}
  shows ¬(T{is linearly independent})
proof
  assume T{is linearly independent}
  then have reg:∀n∈nat. ∀AA∈n→R. ∀B∈inj(n,T).(∑[n;{AA,B}] =Θ ) ⟶
(∀m∈n. AAm=0)
    unfolding LinInde_def[OF assms(2)] by auto
  from assms(3) obtain r where r:r∈R r≠0 using Ring_ZF_1_L2(1) by auto
  let A={⟨0,r⟩}
  let B={⟨0,Θ⟩}
  have A:A:succ(0)→R using `r∈R` unfolding Pi_def function_def domain_def
by auto
  have B:B:succ(0)→T using assms(1) unfolding Pi_def function_def domain_def
by auto
  with assms(2) have B2:B:succ(0)→ℳ unfolding Pi_def by auto
  have C:succ(0)∈nat by auto
  have fff:{⟨m, (A m) ·S (B m)⟩ . m ∈ 1}:1→ℳ using coordinate_function[OF
A B2] by auto
  have B∈inj(succ(0),T) unfolding inj_def using apply_equality B by auto
  with A C reg have ∑[succ(0);{A,B}] =Θ ⟶ (∀m∈succ(0). Am=0) by blast
  then have ∑[succ(0);{A,B}] =Θ ⟶ (A0=0) by blast
  then have ∑[succ(0);{A,B}] =Θ ⟶ (r=0) using apply_equality[OF _
A,of 0 r] by auto
  with r(2) have ∑[succ(0);{A,B}] ≠Θ by auto
  moreover have ∑[succ(0);{A,B}]=(A0) ·S (B 0) using linComb_one_element[OF
_ A B2] unfolding succ_def by auto
  also have ...=r·SΘ using apply_equality A B by auto
  also have ...=Θ using zero_fixed r(1) by auto
  ultimately show False by auto
qed
```

52.4 Submodule

A submodule is a subgroup that is invariant by the action

```
definition(in module0)
  IsAsubmodule
  where IsAsubmodule(𝒩) ≡ (∀r∈R. ∀h∈𝒩. r·S h∈𝒩) ∧ IsAsubgroup(𝒩,AM)

lemma(in module0) sumodule_is_subgroup:
  assumes IsAsubmodule(𝒩)
  shows IsAsubgroup(𝒩,AM)
  using assms unfolding IsAsubmodule_def by auto

lemma(in module0) sumodule_is_subaction:
  assumes IsAsubmodule(𝒩) r∈R h∈𝒩
  shows r·S h∈𝒩
  using assms unfolding IsAsubmodule_def by auto
```

For groups, we need to prove that the inverse function is closed in a set to prove that set to be a subgroup. In module, that is not necessary.

```
lemma(in module0) inverse_in_set:
  assumes ∀r∈R. ∀h∈𝒩. r·S h∈𝒩 𝒩⊆ℳ
  shows ∀h∈𝒩. (−h)∈𝒩
proof
  fix h assume h∈𝒩 moreover
  then have h∈ℳ using assms(2) by auto
  then have (-1)·S h=(−h) using inv_module by auto moreover
  have (-1)∈R using Ring_ZF_1_L2(2) Ring_ZF_1_L3(1) by auto ultimately
  show (−h)∈𝒩 using assms(1) by force
qed

corollary(in module0) submoduleI:
  assumes 𝒩⊆ℳ 𝒩≠0 𝒩{is closed under}AM  ∀r∈R. ∀h∈𝒩. r·S h∈𝒩
  shows IsAsubmodule(𝒩) unfolding IsAsubmodule_def
  using inverse_in_set[OF assms(4,1)] assms mod_ab_gr.group0_3_T3
    by auto
```

Every module has at least two submodules: the whole module and the trivial module.

```
corollary(in module0) trivial_submodules:
  shows IsAsubmodule(ℳ) and IsAsubmodule({Θ})
  unfolding IsAsubmodule_def
proof(safe)
  have AM:ℳ×ℳ→ℳ using mod_ab_gr.group_oper_fun by auto
  then have AM⊆((ℳ×ℳ)×ℳ) unfolding Pi_def by auto
  then have restrict(AM,ℳ×ℳ)=AM unfolding restrict_def by blast
  then show IsAsubgroup(ℳ,AM) using mAbGr unfolding IsAsubgroup_def
by auto
next
```

```
  fix r h assume A:r∈R h∈𝓜
  from A(1) have Hr:𝓜→𝓜 using H_val_type(2) by auto
  with A(2) show r·_S h∈𝓜 using apply_type[of Hr 𝓜 λt. 𝓜] by auto
next
  fix r assume r∈R
  with zero_fixed show r ·_S Θ = Θ by auto
next
  have {Θ}≠0 by auto moreover
  have {Θ}⊆𝓜 using mod_ab_gr.group0_2_L2 by auto moreover
  {
    fix x y assume x∈{Θ} y∈{Θ}
    then have A_M⟨x,y⟩=Θ using mod_ab_gr.group0_2_L2 by auto
  }
  then have {Θ}{is closed under}A_M unfolding IsOpClosed_def by auto
moreover
  {
    fix x assume x∈{Θ}
    then have GroupInv(𝓜, A_M) (x)= Θ using mod_ab_gr.group_inv_of_one
by auto
  }
  then have ∀x∈{Θ}. GroupInv(𝓜, A_M) (x)∈{Θ} by auto ultimately
  show IsAsubgroup({Θ},A_M) using mod_ab_gr.group0_3_T3 by auto
qed
```

The restriction of the action is an action.

```
lemma(in module0) action_submodule:
  assumes IsAsubmodule(𝓝)
  shows {⟨r,restrict(Hr,𝓝)⟩. r∈R}:R→End(𝓝,restrict(A_M,𝓝×𝓝))
proof-
  have sub:𝓝⊆𝓜 using mod_ab_gr.group0_3_L2[OF sumodule_is_subgroup[OF
assms]] by auto
  {
    fix t assume t∈{⟨r,restrict(Hr,𝓝)⟩. r∈R}
    then obtain r where t:r∈R t=⟨r,restrict(Hr,𝓝)⟩ by auto
    then have E:Hr∈End(𝓜,A_M) using H_val_type(1) by auto
    from t(1) have Hr:𝓜→𝓜 using H_val_type(2) by auto
    then have restrict(Hr,𝓝):𝓝→𝓜 using restrict_fun sub by auto more-
over
    have ∀h∈𝓝. restrict(Hr,𝓝)h∈𝓝 using restrict sumodule_is_subaction[OF
assms `r∈R`] by auto
    ultimately have HH:restrict(Hr,𝓝):𝓝→𝓝 using func1_1_L1A by auto
    {
      fix g1 g2 assume H:g1∈𝓝g2∈𝓝
      with sub have G:g1∈𝓜g2∈𝓜 by auto
      from H have AA:A_M⟨g1,g2⟩=restrict(A_M,𝓝×𝓝)⟨g1,g2⟩ using restrict
by auto
      then have (Hr)(A_M⟨g1,g2⟩)=(Hr)(restrict(A_M,𝓝×𝓝)⟨g1,g2⟩) by auto
      then have A_M⟨(Hr)g1,(Hr)g2⟩=(Hr)(restrict(A_M,𝓝×𝓝)⟨g1,g2⟩) us-
ing E G
```

```
        unfolding End_def Homomor_def IsMorphism_def by auto
      with H have restrict(A_M,𝒩×𝒩)⟨(Hr)g1,(Hr)g2⟩=(Hr)(restrict(A_M,𝒩×𝒩)⟨g1,g2⟩)
using sumodule_is_subaction[OF assms `r∈R`]
        by auto moreover
      from H have A_M⟨g1,g2⟩∈𝒩 using sumodule_is_subgroup[OF assms] mod_ab_gr.group0_3_L6
by auto
      then have restrict(Hr,𝒩)(A_M⟨g1,g2⟩)=(Hr)(A_M⟨g1,g2⟩) by auto
      with AA have restrict(Hr,𝒩)(restrict(A_M,𝒩×𝒩)⟨g1,g2⟩)=(Hr)(restrict(A_M,𝒩×𝒩)⟨g1,g
by auto moreover
      from H have (Hr)g1=restrict(Hr,𝒩)g1(Hr)g2=restrict(Hr,𝒩)g2 by
auto ultimately
      have restrict(Hr,𝒩)(restrict(A_M,𝒩×𝒩)⟨g1,g2⟩) = restrict(A_M,𝒩×𝒩)⟨restrict(Hr,𝒩)g
by auto
    }
    then have ∀g1∈𝒩. ∀g2∈𝒩. restrict(Hr,𝒩)(restrict(A_M,𝒩×𝒩)⟨g1,g2⟩)
= restrict(A_M,𝒩×𝒩)⟨restrict(Hr,𝒩)g1,restrict(Hr,𝒩)g2⟩ by auto
    then have Homomor(restrict(Hr,𝒩),𝒩,restrict(A_M,𝒩×𝒩),𝒩,restrict(A_M,𝒩×𝒩))
using HH
      unfolding Homomor_def IsMorphism_def by auto
    with HH have restrict(Hr,𝒩)∈End(𝒩,restrict(A_M,𝒩×𝒩)) unfolding
End_def by auto
    then have t∈R×End(𝒩,restrict(A_M,𝒩×𝒩)) using t by auto
  }
  then have {⟨r,restrict(Hr,𝒩)⟩. r∈R}⊆R×End(𝒩,restrict(A_M,𝒩×𝒩))
by auto moreover
  {
    fix x y assume ⟨x,y⟩∈{⟨r,restrict(Hr,𝒩)⟩. r∈R}
    then have y:x∈Ry=restrict(Hx,𝒩) by auto
    {
      fix y' assume ⟨x,y'⟩∈{⟨r,restrict(Hr,𝒩)⟩. r∈R}
      then have y'=restrict(Hx,𝒩) by auto
      with y(2) have y=y' by auto
    }
    then have ∀y'. ⟨x,y'⟩∈{⟨r,restrict(Hr,𝒩)⟩. r∈R} ⟶ y=y' by auto
  }
  then have ∀x y. ⟨x,y⟩∈{⟨r,restrict(Hr,𝒩)⟩. r∈R} ⟶ (∀y'. ⟨x,y'⟩∈{⟨r,restrict(Hr,𝒩)⟩.
r∈R} ⟶ y=y') by auto
  moreover
  have domain({⟨r,restrict(Hr,𝒩)⟩. r∈R})⊆R unfolding domain_def by auto
  ultimately show fun:{⟨r,restrict(Hr,𝒩)⟩. r∈R}:R→End(𝒩,restrict(A_M,𝒩×𝒩))
unfolding Pi_def function_def by auto
qed
```

A submodule is a module with the restricted action.

```
corollary(in module0) submodule:
  assumes IsAsubmodule(𝒩)
  shows IsLeftModule(R,A,M,𝒩,restrict(A_M,𝒩×𝒩),{⟨r,restrict(Hr,𝒩)⟩.
r∈R})
  unfolding IsLeftModule_def IsAction_def ringHomomor_def IsMorphism_def
```

```
proof(safe)
  show IsAring(R, A, M) using ringAssum by auto
  show g:IsAgroup(𝒩, restrict(A_M,𝒩×𝒩)) using sumodule_is_subgroup
assms unfolding IsAsubgroup_def
    by auto
  have sub:𝒩⊆ℳ using mod_ab_gr.group0_3_L2[OF sumodule_is_subgroup[OF
assms]] by auto
  then show restrict(A_M,𝒩×𝒩) {is commutative on} 𝒩 using mAbGr
    using func_ZF_4_L1[OF mod_ab_gr.group_oper_fun] by auto
  show fun:{⟨r,restrict(Hr,𝒩)⟩. r∈R}:R→End(𝒩,restrict(A_M, 𝒩 × 𝒩))
using action_submodule assms by auto
  then have Q:{⟨r,restrict(Hr,𝒩)⟩. r∈R}1=restrict(H1,𝒩) using apply_equality
Ring_ZF_1_L2(2)
    by auto
  then have ∀h∈𝒩. ({⟨r,restrict(Hr,𝒩)⟩. r∈R}1)h=restrict(H1,𝒩)h by
auto
  then have ∀h∈𝒩. ({⟨r,restrict(Hr,𝒩)⟩. r∈R}1)h=(H1)h using restrict
by auto
  then have ∀h∈𝒩. ({⟨r,restrict(Hr,𝒩)⟩. r∈R}1)h=h using module_ax4
    sub by auto
  then have ∀h∈𝒩. ({⟨r,restrict(Hr,𝒩)⟩. r∈R}1)h=id(𝒩)h by auto more-
over
  have id(𝒩):𝒩→𝒩 using id_inj unfolding inj_def by auto moreover
  from Q have {⟨r,restrict(Hr,𝒩)⟩. r∈R}1:𝒩→ℳ using restrict_fun[OF
H_val_type(2)[OF Ring_ZF_1_L2(2)] sub]
    by auto
  ultimately have {⟨r,restrict(Hr,𝒩)⟩. r∈R}1=id(𝒩) using fun_extension[of
{⟨r,restrict(Hr,𝒩)⟩. r∈R}1 𝒩 λ_. ℳ id(𝒩)] by auto
  also have ...=TheNeutralElement(End(𝒩, restrict(A_M, 𝒩 × 𝒩)), restrict(Composition(𝒩),
End(𝒩, restrict(A_M, 𝒩 × 𝒩)) × End(𝒩, restrict(A_M, 𝒩 × 𝒩))))
    using group0.end_comp_monoid(2) sumodule_is_subgroup[OF assms] un-
folding group0_def IsAsubgroup_def
    by auto
  ultimately show {⟨r,restrict(Hr,𝒩)⟩. r∈R}TheNeutralElement(R, M)=TheNeutralElement(End(𝒩,
restrict(A_M, 𝒩 × 𝒩)),   EndMult(𝒩, restrict(A_M, 𝒩 × 𝒩)))
    unfolding EndMult_def  by auto
  fix r s assume AS:r∈Rs∈R
  then have END:restrict(H  r, 𝒩)∈End(𝒩,restrict(A_M, 𝒩 × 𝒩))restrict(H
s, 𝒩)∈End(𝒩,restrict(A_M, 𝒩 × 𝒩)) using apply_type[OF fun]
    apply_equality[OF _ fun] by auto
  then have funf:restrict(H  r, 𝒩):𝒩→𝒩restrict(H  s, 𝒩):𝒩→𝒩 un-
folding End_def by auto
  from AS have rs:r+s∈Rr·s∈R using Ring_ZF_1_L4(1,3) by auto
  then have EE:{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (r+s)=restrict(H  (r+s),
𝒩) using apply_equality fun by auto
  have m:monoid0(𝒩,restrict(A_M, 𝒩 × 𝒩)) unfolding monoid0_def
    using g unfolding IsAgroup_def by auto
  have f1:{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (r+s):𝒩→𝒩 using apply_type[OF
fun rs(1)] unfolding End_def by auto
```

```
  have f:(restrict(A_M, 𝒩 × 𝒩) {lifted to function space over} 𝒩)
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩:𝒩→𝒩 using monoid0.Group_ZF_2_1_L0[OF m _ funf]
          AS apply_equality[OF _ fun] by auto
  from END have ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s⟩ ∈ End(𝒩, restrict(A_M, 𝒩 × 𝒩)) × End(𝒩, restrict(A_M,
𝒩 × 𝒩))
       using apply_equality[OF _ fun] AS by auto
  from apply_type[OF restrict_fun[OF monoid0.Group_ZF_2_1_L0A[OF m, of
restrict(A_M,𝒩×𝒩) {lifted to function space over}𝒩 𝒩], of End(𝒩,restrict(A_M,
𝒩 × 𝒩))×End(𝒩,restrict(A_M, 𝒩 × 𝒩))] this]
     have f2:(EndAdd(𝒩, restrict(A_M, 𝒩 × 𝒩))
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩):𝒩→𝒩
      unfolding EndAdd_def End_def by blast
  {
    fix g assume gh:g∈𝒩
    have A:((restrict(A_M, 𝒩 × 𝒩)  {lifted to function space over} 𝒩)

          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩)g=((restrict(A_M, 𝒩 × 𝒩) {lifted to function space over}
𝒩)
          ⟨restrict(H  r, 𝒩), restrict(H  s, 𝒩)⟩)g using apply_equality[OF
_ fun] AS by auto
    from END have ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s⟩ ∈ End(𝒩, restrict(A_M, 𝒩 × 𝒩)) × End(𝒩, restrict(A_M,
𝒩 × 𝒩))
       using apply_equality[OF _ fun] AS by auto
     with A have (EndAdd(𝒩, restrict(A_M, 𝒩 × 𝒩))
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩)g=((restrict(A_M, 𝒩 × 𝒩) {lifted to function space over}
𝒩)
          ⟨restrict(H  r, 𝒩), restrict(H  s, 𝒩)⟩)g unfolding EndAdd_def

       using restrict[of (restrict(A_M, 𝒩 × 𝒩)  {lifted to function
space over} 𝒩) End(𝒩,restrict(A_M, 𝒩 × 𝒩))×End(𝒩,restrict(A_M, 𝒩
× 𝒩)) ⟨restrict(H  r, 𝒩), restrict(H  s, 𝒩)⟩]
      by auto
    also have ...=restrict(A_M,𝒩 × 𝒩)⟨restrict(H  r, 𝒩)g, restrict(H
s, 𝒩)g⟩ using group0.Group_ZF_2_1_L3[OF _ _ funf gh] g
      unfolding group0_def by auto
    also have ...=A_M⟨restrict(H  r, 𝒩)g, restrict(H  s, 𝒩)g⟩ using apply_type[OF
funf(1) gh] apply_type[OF funf(2) gh]
      by auto
    also have ...=A_M⟨(H  r)g, (H  s)g⟩ using gh by auto
    also have ...=(H(r+s))g using module_ax2 AS gh sub by auto
    also have ...=restrict(H(r+s),𝒩)g using gh by auto
    also have ...=({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (r+s))g using EE
by auto
```

```
    ultimately have (EndAdd(𝒩, restrict(A_M, 𝒩 × 𝒩))
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩)g=({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (r+s))g by auto
  }
  then have ∀g∈𝒩. (EndAdd(𝒩, restrict(A_M, 𝒩 × 𝒩))
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩)g=({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (r+s))g by auto
  then show {⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (A  ⟨r, s⟩) =
          EndAdd(𝒩, restrict(A_M, 𝒩 × 𝒩))
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩ using fun_extension[OF f1 f2] by auto
  have f1:({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R} (r·s)):𝒩→𝒩 using apply_type[OF
fun rs(2)] unfolding End_def by auto
  have ff1:{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r:𝒩→𝒩{⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s:𝒩→𝒩 using apply_type[OF fun] AS unfolding End_def
by auto
  then have ({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r)O({⟨r, restrict(H  r,
𝒩)⟩ . r ∈ R}  s):𝒩→𝒩 using comp_fun by auto
  then have f:(Composition(𝒩)  ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r,
{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  s⟩):𝒩→𝒩 using func_ZF_5_L2
    [OF ff1] using EndMult_def by auto
  from END have ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s⟩ ∈ End(𝒩, restrict(A_M, 𝒩 × 𝒩)) × End(𝒩, restrict(A_M,
𝒩 × 𝒩))
      using apply_equality[OF _ fun] AS by auto
  from apply_type[OF restrict_fun[OF func_ZF_5_L1[of 𝒩], of End(𝒩,restrict(A_M,
𝒩 × 𝒩))×End(𝒩,restrict(A_M, 𝒩 × 𝒩))] this]
     have f2:(EndMult(𝒩, restrict(A_M, 𝒩 × 𝒩))
          ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  s⟩):𝒩→𝒩
      unfolding EndMult_def End_def by blast
  {
    fix g assume gh:g∈𝒩
    have A:(Composition(𝒩)  ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r,
restrict(H  r, 𝒩)⟩ . r ∈ R}  s⟩)g=
      (Composition(𝒩)  ⟨restrict(H  r, 𝒩), restrict(H  s,𝒩)⟩)g using
apply_equality[OF _ fun] AS by auto
    from END have ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s⟩ ∈ End(𝒩, restrict(A_M, 𝒩 × 𝒩)) × End(𝒩, restrict(A_M,
𝒩 × 𝒩))
      using apply_equality[OF _ fun] AS by auto
     with A have (EndMult(𝒩, restrict(A_M, 𝒩 × 𝒩))  ⟨{⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  s⟩)g=
      (Composition(𝒩)  ⟨restrict(H  r, 𝒩), restrict(H  s,𝒩)⟩)g
      using restrict unfolding EndMult_def by auto
    also have ...=(restrict(H  r, 𝒩)O restrict(H  s,𝒩))g using func_ZF_5_L2[OF
funf] by auto
    also have ...=restrict(H  r, 𝒩)( restrict(H s,𝒩)g) using comp_fun_apply[OF
funf(2) gh] by auto
```

```
    also have ...=(H  r)((H  s)g) using gh apply_type[OF funf(2) gh] by
auto
    also have ...=(H(r·s))g using module_ax3 gh sub AS by auto
    also have ...=restrict(H  (r·s), 𝒩)g using gh by auto
    also have ...=({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R} (r·s))g using apply_equality[OF
_ fun] rs(2) by auto
    ultimately have ({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R} (r·s))g =(EndMult(𝒩,
restrict(A_M, 𝒩 × 𝒩))  ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s⟩)g
      by auto
  }
  then have ∀g∈𝒩. ({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R} (r·s))g =(EndMult(𝒩,
restrict(A_M, 𝒩 × 𝒩))  ⟨{⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  r, {⟨r, restrict(H
r, 𝒩)⟩ . r ∈ R}  s⟩)g by auto
  then show {⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (M  ⟨r, s⟩) =
          EndMult(𝒩, restrict(A_M, 𝒩 × 𝒩))  ⟨{⟨r, restrict(H  r, 𝒩)⟩
. r ∈ R}  r, {⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  s⟩ using fun_extension[OF
f1 f2] by auto
qed
```

If we consider linear combinations of elements in a submodule, then the linear combination is also in the submodule.

```
lemma(in module0) linear_comb_submod:
  assumes IsAsubmodule(𝒩) D∈FinPow(X) AA:X→R B:X→𝒩
  shows ∑ [D;{AA,B}]∈𝒩
proof-
  have fun:A_M:ℳ×ℳ→ℳ using mod_ab_gr.group_oper_fun.
  from assms(4) have BB:B:X→ℳ using mod_ab_gr.group0_3_L2[OF sumodule_is_subgroup[OF
assms(1)]] func1_1_L1B by auto
  {
    fix A1 B1 assume fun:A1:X→R B1:X→𝒩
    from fun(2) have fun2:B1:X→ℳ using mod_ab_gr.group0_3_L2[OF sumodule_is_subgroup[OF
assms(1)]] func1_1_L1B by auto
    have 0∈FinPow(X) unfolding FinPow_def by auto
    then have ∑ [0;{A1,B1}]=Θ using LinearComb_def[OF fun(1) fun2] by
auto
    then have ∑ [0;{A1,B1}]∈𝒩 using assms(1) mod_ab_gr.group0_3_L5 un-
folding IsAsubmodule_def by auto
  }
  then have base:∀A1∈X→R. ∀B1∈X→𝒩. ∑ [0;{A1,B1}]∈𝒩 by auto
  {
    fix RR assume a:RR≠0 RR∈FinPow(X)
    then obtain d where d:d∈RR by auto
    {
      fix A1 B1 assume fun:A1:X→R B1:X→𝒩 and step:∀A1∈X→R. ∀B1∈X→𝒩.
∑ [RR-{d};{A1,B1}]∈𝒩
      have F:{⟨m, ({⟨r, restrict(H  r, 𝒩)⟩ . r ∈ R}  (A1  m))  (B1  m)⟩
. m ∈ X} : X → 𝒩 using module0.coordinate_function[OF _ fun]
        submodule[OF assms(1)] unfolding module0_def IsLeftModule_def
```

ring0_def module0_axioms_def **by** auto
{
fix m **assume** mx:m∈X
then have bh:B1m∈$\mathcal{N}$ **using** fun(2) apply_type **by** auto
from mx **have** ar:A1m∈R **using** fun(1) apply_type **by** auto
then have A:{⟨r, restrict(H r, $\mathcal{N}$)⟩ . r ∈ R}(A1m)=restrict(H (A1m), $\mathcal{N}$) **using** apply_equality action_submodule[OF assms(1)] **by** auto
have B:(restrict(H (A1m), $\mathcal{N}$)) (B1 m)=(H (A1m))(B1 m) **using** bh restrict **by** auto
with A **have** ({⟨r, restrict(H r, $\mathcal{N}$)⟩ . r ∈ R} (A1 m)) (B1 m)=(H (A1m))(B1 m) **by** auto
}
then have eq1:{⟨m, ({⟨r, restrict(H r, $\mathcal{N}$)⟩ . r ∈ R} (A1 m)) (B1 m)⟩ . m ∈ X} ={⟨m, (H (A1m))(B1 m)⟩ . m ∈ X} **by** auto
then have pd:{⟨m, (H (A1m))(B1 m)⟩ . m ∈ X}d∈$\mathcal{N}$ **using** d apply_type[OF F] a(2) **unfolding** FinPow_def **by** auto
from fun(2) **have** fun2:B1:X→$\mathcal{M}$ **using** mod_ab_gr.group0_3_L2[OF sumodule_is_subgroup[OF assms(1)]] func1_1_L1B **by** auto
have $\sum$[RR;{A1,B1}]=($\sum$[RR-{d};{A1,B1}])$+_V$({⟨k, (A1 k) $\cdot_S$ (B1 k)⟩ . k ∈ X} d) **using** sum_one_element[OF fun(1) fun2 a(2) d].
also have ...∈$\mathcal{N}$ **using** pd step fun mod_ab_gr.group0_3_L6[OF sumodule_is_subgroup[OF assms(1)]] **by** auto
ultimately have $\sum$[RR;{A1,B1}]∈$\mathcal{N}$ **by** auto
}
then have (∀AA1∈X→R. ∀BB1∈X→$\mathcal{N}$. $\sum$[RR-{d};{AA1,BB1}]∈$\mathcal{N}$)⟶(∀AA1∈X→R. ∀BB1∈X→$\mathcal{N}$. $\sum$[RR;{AA1,BB1}]∈$\mathcal{N}$) **by** auto
with d **have** ∃d∈RR. ((∀AA1∈X→R. ∀BB1∈X→$\mathcal{N}$. $\sum$[RR-{d};{AA1,BB1}]∈$\mathcal{N}$)⟶(∀AA1∈X→R. ∀BB1∈X→$\mathcal{N}$. $\sum$[RR;{AA1,BB1}]∈$\mathcal{N}$)) **by** auto
}
then have step:∀RR∈FinPow(X). RR≠0 ⟶ (∃d∈RR. ((∀AA1∈X→R. ∀BB1∈X→$\mathcal{N}$. $\sum$[RR-{d};{AA1,BB1}]∈$\mathcal{N}$)⟶(∀AA1∈X→R. ∀BB1∈X→$\mathcal{N}$. $\sum$[RR;{AA1,BB1}]∈$\mathcal{N}$))) **by** auto
show thesis **using** FinPow_ind_rem_one[OF base step] assms(2-4) **by** auto
qed

52.4.1 Spans

Since we know linear combinations, we can define the span of a subset of a module as the linear combinations of elements in that subset. We have already proven that the sum can be done only over finite numbers considering a bijection between a finite number and the original finite set, and that the function can be restricted to that finite number.

The terms of a linear combination can be reordered so that they are indexed by the elements of the module.

lemma(in module0) index_module:
assumes AAA:X→R BB:X→$\mathcal{M}$ D∈FinPow(X)

```
  shows ∃AA∈ℳ→R. ∑[D;{AAA,BB}]=∑[BBD;{AA,id(ℳ)}] ∧ (∀x∈ℳ-BBD.
AAx=0)
proof-
  let F={⟨d,CommSetFold(A,AAA,D∩(BB-({BBd})))⟩. d∈D}
  let f1={⟨d,D∩(BB-({BBd}))⟩. d∈D}
  have f1:D→{D∩(BB-({BBd})). d∈D} unfolding Pi_def function_def by auto
  then have RepFun(D,λt. f1t)={D∩(BB-({BBd})). d∈D} using apply_equality
by auto
  with assms(3) have Finite({D∩(BB-({BBd})). d∈D}) using Finite_RepFun
unfolding FinPow_def by auto
  then obtain n where {D∩(BB-({BBd})). d∈D}≈n and n:n∈nat unfolding
Finite_def by auto
  then have n2:{D∩(BB-({BBd})). d∈D}≲n using eqpoll_imp_lepoll by auto
  {
    fix T assume T∈{D∩(BB-({BBd})). d∈D}
    then obtain d where d:d∈D T=D∩(BB-({BBd})) by auto
    {
      assume d∉T
      with d have d∉(BB-({BBd})) by auto
      then have ⟨d,BBd⟩∉BB using vimage_iff by auto
      with d(1) assms(2,3) have False unfolding FinPow_def Pi_def us-
ing function_apply_Pair[of BB d] by auto
    }
    then have T≠0 by auto
  }
  then have n3:∀t∈{D∩(BB-({BBd})). d∈D}. id({D∩(BB-({BBd})). d∈D})
t ≠ 0 using id_def by auto
  from n have ∀M N. M ≲ n ∧ (∀t∈M. N  t ≠ 0) ⟶ (∃f. f ∈ Pi(M,λt.
N  t) ∧ (∀t∈M. f  t ∈ N  t)) using finite_choice[of n] unfolding AxiomCardinalChoiceGen_def
    by auto
  with n2 have ∀N. (∀t∈{D∩(BB-({BBd})). d∈D}. N  t ≠ 0) ⟶ (∃f. f
∈ Pi({D∩(BB-({BBd})). d∈D},λt. N  t) ∧ (∀t∈{D∩(BB-({BBd})). d∈D}. f
t ∈ N  t))
    by blast
  with n3 have ∃f. f ∈ Pi({D∩(BB-({BBd})). d∈D},λt. id({D∩(BB-({BBd})).
d∈D})  t) ∧ (∀t∈{D∩(BB-({BBd})). d∈D}. f  t ∈ id({D∩(BB-({BBd})). d∈D})
t)
    by auto
  then obtain ff where ff:ff∈Pi({D∩(BB-({BBd})). d∈D},λt. id({D∩(BB-({BBd})).
d∈D})  t) (∀t∈{D∩(BB-({BBd})). d∈D}. ff  t ∈ id({D∩(BB-({BBd})). d∈D})
t) by force
  {
    fix t assume as:t∈{D∩(BB-({BBd})). d∈D}
    with ff(2) have fft∈id({D∩(BB-({BBd})). d∈D})  t by blast
    with as have fft∈t using id_def by auto
  }
  then have ff2:∀t∈{D∩(BB-({BBd})). d∈D}. ff  t ∈t by auto
  have ∀x∈{D∩(BB-({BBd})). d∈D}. id({D∩(BB-({BBd})). d∈D})x=x using
id_def by auto
```

```
  with ff(1) have ff1:ff∈Pi({D∩(BB-({BBd})). d∈D},λt. t) unfolding Pi_def
Sigma_def by auto
  have case0:∃AA∈ℳ→R. ∑[0;{AAA,BB}]=∑[BB0;{AA,id(ℳ)}] ∧ (∀x∈ℳ-BB0.
AAx=0)
    proof
      have ∑[0;{AAA,BB}]=Θ using LinearComb_def[OF assms(1,2)] unfold-
ing FinPow_def by auto moreover
      let A=ConstantFunction(ℳ,0)
      have ∑[0;{A,id(ℳ)}]=Θ using LinearComb_def[OF func1_3_L1[OF Ring_ZF_1_L2(1)],
of id(ℳ) ℳ 0]
        unfolding id_def FinPow_def by auto moreover
      have BB0=0 by auto ultimately
      show ∑[0;{AAA,BB}]=∑[BB0;{A,id(ℳ)}] ∧ (∀x∈ℳ-BB0. Ax=0) us-
ing func1_3_L2 by auto
      then show A:ℳ→R using func1_3_L1[OF Ring_ZF_1_L2(1), of ℳ] by
auto
    qed
  {
    fix E assume E:E∈FinPow(X) E≠0
    {
      fix d assume d:d∈E
      {
        assume hyp:∃AA∈ℳ→R. ∑[E-{d};{AAA,BB}]=∑[BB(E-{d});{AA,id(ℳ)}]
∧ (∀x∈ℳ-BB(E-{d}). AAx=0)
        from hyp obtain AA where AA:AA∈ℳ→R ∑[E-{d};{AAA,BB}]=∑[BB(E-{d});{AA,id(ℳ)}]
∀x∈ℳ-BB(E-{d}). AAx=0 by auto
        have ∑[E;{AAA,BB}] = (∑[E-{d};{AAA,BB}])+V({⟨e,(AAAe)·S(BBe)⟩.
e∈X}d) using sum_one_element[OF assms(1,2) E(1) d].
        with AA(2) also have ...=(∑[BB(E-{d});{AA,id(ℳ)}])+V((AAAd)·S(BBd))
using d E(1) unfolding FinPow_def using apply_equality[OF _
          coordinate_function[OF assms(1,2)]] by auto
        ultimately have eq:∑[E;{AAA,BB}] =(∑[BB(E-{d});{AA,id(ℳ)}])+V((AAAd)·S(BBd))
by auto
        have btype:BBd∈ℳ using apply_type assms(2) d E(1) unfolding
FinPow_def by auto
        have (E-{d})∈FinPow(X) using E(1) unfolding FinPow_def using subset_Finite[of
E-{d} E] by auto moreover
        then have {BBx. x∈E-{d}}=BB(E-{d}) using func_imagedef[OF assms(2),
of E-{d}] unfolding FinPow_def
          by auto
        ultimately have fin:Finite(BB(E-{d})) using Finite_RepFun[of E-{d}
λt. BBt] unfolding FinPow_def by auto
        then have Finite(BB(E-{d})∪{BBd}) using Finite_cons[of BB(E-{d})
BBd] by auto
        with btype have finpow:BB(E-{d})∪{BBd}∈FinPow(ℳ) using func1_1_L6(2)[OF
assms(2)] unfolding FinPow_def by auto
        {
          assume as:BBd∉BB(E-{d})
          then have T_def:BB(E-{d})=(BB(E-{d})∪{BBd})-{BBd} by auto
```

from as have sub:BB(E-{d})⊆ℳ-{BBd} using func1_1_L6(2)[OF assms(2)] by auto
let A=restrict(AA,ℳ-{BBd})∪ConstantFunction({BBd},AAAd)
have res:restrict(AA,ℳ-{BBd}):ℳ-{BBd}→R using restrict_fun[OF AA(1)] by auto
moreover have AAAd∈R using apply_type assms(1) d E(1) unfolding FinPow_def by auto
then have con:ConstantFunction({BBd},AAAd):{BBd}→R using func1_3_L1 by auto
moreover have (G-{BBd})∩{BBd}=0 by auto moreover
have R∪R=R by auto moreover
have (ℳ-{BBd})∪{BBd}=ℳ using apply_type[OF assms(2)] d E(1) unfolding FinPow_def by auto ultimately
have A_fun:A:ℳ→R using fun_disjoint_Un[of restrict(AA,ℳ-{BBd}) ℳ-{BBd} R ConstantFunction({BBd},AAAd) {BBd} R] by auto
have A(BBd)=ConstantFunction({BBd},AAAd)(BBd) using as fun_disjoint_apply2 by auto moreover note btype
ultimately have A_app:A(BBd)=AAAd using as func1_3_L2 by auto
{
fix z assume z∈restrict(A,BB(E-{d}))
then have z:z∈A ∃x∈BB (E - {d}). ∃y. z = ⟨x, y⟩ using restrict_iff by auto
then have ∃x∈BB (E - {d}). ∃y. z = ⟨x, y⟩ z∈ConstantFunction({BBd},AAAd) ∨ z∈restrict(AA,ℳ-{BBd}) by auto
then have ∃x∈BB (E - {d}). ∃y. z = ⟨x, y⟩ z∈{BBd}×{AAAd} ∨ z∈restrict(AA,ℳ-{BBd}) using ConstantFunction_def by auto
then have fst(z)∈BB (E - {d}) z=⟨BBd,AAAd⟩ ∨ z∈restrict(AA,ℳ-{BBd}) by auto
with as have z∈restrict(AA,ℳ-{BBd}) by auto
with z(2) have z∈AA ∃x∈BB (E - {d}). ∃y. z = ⟨x, y⟩ using restrict_iff by auto
then have z∈restrict(AA,BB(E-{d})) using restrict_iff by auto
}
then have restrict(A,BB(E-{d}))⊆restrict(AA,BB(E-{d})) by auto moreover
{
fix z assume z:z∈restrict(AA,BB(E-{d})) z∉restrict(A,BB(E-{d}))
then have disj:z∉A ∨ (∀x∈BB(E-{d}). ∀y. z ≠ ⟨x, y⟩) using restrict_iff[of z A BB(E-{d})] by auto moreover
with z(1) have z:z∈AA ∃x∈BB(E-{d}). ∃y. z=⟨x, y⟩ using restrict_iff[of _ AA BB(E-{d})] by auto moreover
from z(2) sub have ∃x∈ℳ-{BBd}. ∃y. z=⟨x, y⟩ by auto
with z(1) have z∈restrict(AA,ℳ-{BBd}) using restrict_iff by auto
then have z∈A by auto
with disj have ∀x∈BB(E-{d}). ∀y. z ≠ ⟨x, y⟩ by auto
with z(2) have False by auto
}

```
          then have restrict(AA,BB(E-{d}))⊆restrict(A,BB(E-{d})) by auto
ultimately
          have resA:restrict(AA,BB(E-{d}))=restrict(A,BB(E-{d})) by auto
          have ∑[BB(E-{d});{AA,id(ℳ)}]=∑[BB(E-{d});{restrict(AA,BB(E-{d})),restrict(id(
BB(E-{d}))}] using linComb_restrict_coord[OF AA(1) id_type, of BB(E-{d})]
            fin func1_1_L6(2)[OF assms(2)] unfolding FinPow_def by auto
          also have ...=∑[BB(E-{d});{restrict(A,BB(E-{d})),restrict(id(ℳ),
BB(E-{d}))}] using resA by auto
          also have ...=∑[BB(E-{d});{A,id(ℳ)}] using linComb_restrict_coord[OF
A_fun id_type, of BB(E-{d})] fin func1_1_L6(2)[OF assms(2)] unfolding
FinPow_def by auto
          ultimately have ∑[BB(E-{d});{AA,id(ℳ)}]=∑[(BB(E-{d})∪{BBd})-{BBd};{A,id(ℳ)}]
using T_def by auto
          then have (∑[BB(E-{d});{AA,id(ℳ)}])+V((AAAd)·S(BBd))=(∑[(BB(E-{d})∪{BBd})-{BB
using A_app by auto
          also have ...=(∑[(BB(E-{d})∪{BBd})-{BBd};{A,id(ℳ)}])+V((A(BBd))·S(id(ℳ)(BBd)))
using id_conv[OF btype] by auto
          also have ...=(∑[(BB(E-{d})∪{BBd})-{BBd};{A,id(ℳ)}])+V({⟨g,(Ag)·S(id(ℳ)g)⟩.
g∈ℳ}(BBd)) using apply_equality[OF _ coordinate_function[OF A_fun id_type]
            ,of BBd (A(BBd))·S(id(ℳ)(BBd))] btype by auto
          also have ...=(∑[(BB(E-{d})∪{BBd});{A,id(ℳ)}]) using sum_one_element[OF
A_fun id_type finpow, of BBd] by auto
          ultimately have (∑[BB(E-{d});{AA,id(ℳ)}])+V((AAAd)·S(BBd))=∑[(BB(E-{d})∪{BBd}
by auto
          with eq have eq:∑[E;{AAA,BB}] =∑[(BB(E-{d})∪{BBd});{A,id(ℳ)}]
by auto
          have BB(E-{d})∪{BBd}={BBx. x∈E-{d}}∪{BBd} using func_imagedef[OF
assms(2), of E-{d}]
            E(1) unfolding FinPow_def by force
          also have ...={BBx. x∈E} using d by auto ultimately
          have set:BB(E-{d})∪{BBd}=BBE using func_imagedef[OF assms(2),
of E] E(1) unfolding FinPow_def by auto
          {
            fix x assume x:x∈ℳ-BBE
            then have x1:x≠BBd using d func_imagedef[OF assms(2), of
E] E(1) unfolding FinPow_def by auto
            then have Ax=restrict(AA,ℳ-{BBd})x using fun_disjoint_apply1[of
x ConstantFunction({BBd},AAAd)] unfolding
              ConstantFunction_def by blast
            with x x1 have Ax=AAx using restrict by auto moreover
            from x set have x∈ℳ-BB(E-{d}) by auto ultimately
            have Ax=0 using AA(3) by auto
          }
          with set eq A_fun have ∃AA∈ℳ→R. ∑[E;{AAA,BB}]=∑[BBE;{AA,id(ℳ)}]
∧ (∀x∈ℳ-(BBE). AAx=0) by auto
        }
        moreover
        {
          assume as:BBd∈BB(E-{d})
```

```
          then have BB(E-{d})∪{BBd}=BB(E-{d}) by auto
          then have finpow:BB(E-{d})∈FinPow(ℳ) using finpow by auto
          have sub:E-{d}⊆X using E(1) unfolding FinPow_def by force
          with as have BBd∈{BBf. f∈E-{d}} using func_imagedef[OF assms(2),
of E-{d}] by auto
          then have {BBf. f∈E-{d}}={BBf. f∈E} by auto
          then have im_eq:BB(E-{d})=BBE using func_imagedef[OF assms(2),of
E-{d}] func_imagedef[OF assms(2),of E] sub E(1) unfolding FinPow_def
            by auto
          from as have ∑[BB(E-{d});{AA,id(ℳ)}]=(∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V({⟨g,
g∈ℳ}(BBd)) using sum_one_element[OF AA(1) id_type]
            finpow by auto
          also have ...=(∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V((AA(BBd))·S(id(ℳ)(BBd)))
using apply_equality[OF _ coordinate_function[OF AA(1) id_type]]
            btype by auto
          also have ...=(∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V((AA(BBd))·S(BBd))
using id_conv btype by auto
          ultimately have ∑[BB(E-{d});{AA,id(ℳ)}]=(∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V((A
by auto
          then have ∑[E;{AAA,BB}] =((∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V((AA(BBd))·S(BBd))
d)·S (BB  d)) using eq by auto
          moreover have ∑[BB(E-{d})-{BBd};{AA,id(ℳ)}]∈ℳ using linComb_is_in_module[OF
AA(1) id_type, of BB(E-{d})-{BBd}] finpow subset_Finite[of BB(E-{d})-{BBd}
BB(E-{d})] unfolding FinPow_def
            by auto moreover
          have (AA(BBd))·S(BBd)∈ℳ using apply_type[OF AA(1) btype] apply_type[OF
H_val_type(2)]
            btype by auto  moreover
          have (AAAd)·S(BBd)∈ℳ using apply_type apply_type[OF assms(1),
of d] apply_type[OF H_val_type(2)]
             btype d E(1) unfolding FinPow_def by auto ultimately
          have ∑[E;{AAA,BB}] =(∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V(((AA(BBd))·S(BBd))+V((A
d) ·S (BB  d)))
            using mod_ab_gr.group_oper_assoc by auto moreover
          have ((AA(BBd))·S(BBd))+V((AAA  d) ·S (BB  d))=((AA(BBd))+(AAA
d))·S(BBd) using btype apply_type[OF assms(1), of d]
            apply_type[OF AA(1) btype] d E(1) module_ax2 unfolding FinPow_def
by auto
          ultimately have eq:∑[E;{AAA,BB}] =(∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V(((AA(BBd)
d))·S(BBd)) by auto
          let A=restrict(AA,ℳ-{BBd})∪ConstantFunction({BBd},(AA(BBd))+(AAA
d))
          have (ℳ-{BBd})∩({BBd})=0 by auto moreover
          have (ℳ-{BBd})∪({BBd})=ℳ using finpow as unfolding FinPow_def
by auto moreover
          have restrict(AA,ℳ-{BBd}):(ℳ-{BBd})→R using restrict_fun[OF
AA(1), of ℳ-{BBd}] finpow unfolding FinPow_def by auto moreover
          have AAAd∈R AA(BBd)∈R using apply_type assms(1) AA(1) btype
d E(1) unfolding FinPow_def by auto
```

```
          then have (AA(BBd))+(AAA  d)∈R using Ring_ZF_1_L4(1) by auto
          then have ConstantFunction({BBd},(AA(BBd))+(AAA  d)):{BBd}→R
using func1_3_L1 by auto
          ultimately have A_fun:A:𝓜→R using fun_disjoint_Un[of restrict(AA,𝓜-{BBd})
𝓜-{BBd} R ConstantFunction({BBd},(AA(BBd))+(AAA  d)) {BBd} R] by auto
          have A(BBd)=ConstantFunction({BBd},(AA(BBd))+(AAA  d))(BBd)
using as fun_disjoint_apply2 by auto moreover note btype
          ultimately have A_app:A(BBd)=(AA(BBd))+(AAA  d) using as func1_3_L2
by auto
          {
            fix z assume z∈restrict(A,BB(E-{d})-{BBd})
            then have z:∃x∈BB(E-{d})-{BBd}. ∃y. z=⟨x,y⟩ z∈A using restrict_iff
by auto
            then have ∃x∈BB  (E - {d})-{BBd}. ∃y. z = ⟨x, y⟩ z∈ConstantFunction({BBd},(AA(
d)) ∨ z∈restrict(AA,𝓜-{BBd}) by auto
            then have ∃x∈BB  (E - {d})-{BBd}. ∃y. z = ⟨x, y⟩ z∈{BBd}×{(AA(BBd))+(AAA
d)} ∨ z∈restrict(AA,𝓜-{BBd}) using ConstantFunction_def by auto
            then have fst(z)∈BB  (E - {d})-{BBd} z=⟨BBd,AAAd⟩ ∨ z∈restrict(AA,𝓜-{BBd})
by auto
            then have z∈restrict(AA,𝓜-{BBd}) by auto
            with z(1) have z∈AA ∃x∈BB  (E - {d})-{BBd}. ∃y. z = ⟨x,
y⟩ using restrict_iff by auto
            then have z∈restrict(AA,BB(E-{d})-{BBd}) using restrict_iff
by auto
          }
          then have restrict(A,BB(E-{d})-{BBd})⊆restrict(AA,BB(E-{d})-{BBd})
by auto moreover
          {
            fix z assume z:z∈restrict(AA,BB(E-{d})-{BBd}) z∉restrict(A,BB(E-{d})-{BBd})
            then have disj:z∉A ∨ (∀x∈BB(E-{d})-{BBd}. ∀y. z ≠ ⟨x, y⟩)
using restrict_iff[of z A BB(E-{d})-{BBd}] by auto moreover
            with z(1) have z:z∈AA ∃x∈BB(E-{d})-{BBd}. ∃y. z=⟨x, y⟩ us-
ing restrict_iff[of _ AA BB(E-{d})-{BBd}] by auto moreover
            from z(2) func1_1_L6(2)[OF assms(2)] have ∃x∈𝓜-{BBd}. ∃y.
z=⟨x, y⟩ by auto
            with z(1) have z∈restrict(AA,𝓜-{BBd}) using restrict_iff
by auto
            then have z∈A by auto
            with disj have ∀x∈BB(E-{d})-{BBd}. ∀y. z ≠ ⟨x, y⟩ by auto
            with z(2) have False by auto
          }
          then have restrict(AA,BB(E-{d})-{BBd})⊆restrict(A,BB(E-{d})-{BBd})
by auto ultimately
          have resA:restrict(AA,BB(E-{d})-{BBd})=restrict(A,BB(E-{d})-{BBd})
by auto
          have Finite(BB(E-{d})-{BBd}) using finpow unfolding FinPow_def
using subset_Finite[of BB(E-{d}) -{BBd}] by auto
          with finpow have finpow2:BB(E-{d})-{BBd}∈FinPow(𝓜) unfold-
ing FinPow_def by auto
```

```
          have ∑[BB(E-{d})-{BBd};{AA,id(ℳ)}]=∑[BB(E-{d})-{BBd};{restrict(AA,BB(E-{d})-{
BB(E-{d})-{BBd})}] using linComb_restrict_coord[OF AA(1) id_type finpow2]
            by auto
          also have ...=∑[BB(E-{d})-{BBd};{restrict(A,BB(E-{d})-{BBd}),restrict(id(ℳ),
BB(E-{d})-{BBd})}] using resA by auto
          also have ...=∑[BB(E-{d})-{BBd};{A,id(ℳ)}] using linComb_restrict_coord[OF
A_fun id_type finpow2] by auto
          ultimately have ∑[BB(E-{d})-{BBd};{AA,id(ℳ)}]=∑[BB(E-{d})-{BBd};{A,id(ℳ)}]
by auto
          then have (∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V(((AA(BBd))+(AAA
d))·S(BBd))=(∑[BB(E-{d})-{BBd};{A,id(ℳ)}])+V((A(BBd))·S(BBd)) using
A_app by auto
          also have ...=(∑[BB(E-{d})-{BBd};{A,id(ℳ)}])+V((A(BBd))·S(id(ℳ)(BBd)))
using id_conv[OF btype] by auto
          also have ...=(∑[BB(E-{d})-{BBd};{A,id(ℳ)}])+V({⟨g,(Ag)·S(id(ℳ)g)⟩.
g∈ℳ}(BBd)) using apply_equality[OF _ coordinate_function[OF A_fun id_type]
            ,of BBd (A(BBd))·S(id(ℳ)(BBd))] btype by auto
          also have ...=(∑[BB(E-{d});{A,id(ℳ)}]) using sum_one_element[OF
A_fun id_type finpow as] by auto
          ultimately have (∑[BB(E-{d})-{BBd};{AA,id(ℳ)}])+V(((AA(BBd))+(AAA
d))·S(BBd))=(∑[BB(E-{d});{A,id(ℳ)}]) by auto
          with eq have sol:∑[E;{AAA,BB}] =∑[BB(E-{d});{A,id(ℳ)}] by
auto
          {
            fix x assume x:x∈ℳ-BBE
            with as have x1:x∈ℳ-{BBd} by auto
            then have Ax=restrict(AA,ℳ-{BBd})x using fun_disjoint_apply1[of
x ConstantFunction({BBd},(AA(BBd))+(AAA  d))]
              unfolding ConstantFunction_def by blast
            with x1 have Ax=AAx using restrict by auto
            with x im_eq have Ax=0 using AA(3) by auto
          }
          with sol A_fun im_eq have ∃AA∈ℳ→R. ∑[E;{AAA,BB}]=∑[BB(E);{AA,id(ℳ)}]
∧ (∀x∈ℳ-BBE. AAx=0) by auto
        }
        ultimately have ∃AA∈ℳ→R. ∑[E;{AAA,BB}]=∑[BBE;{AA,id(ℳ)}]
∧ (∀x∈ℳ-BBE. AAx=0) by auto
      }
      then have ( ∃AA∈ℳ→R. (∑[E-{d};{AAA,BB}])=(∑[BB(E-{d});{AA,id(ℳ)}])
∧ (∀x∈ℳ-BB(E-{d}). AAx=0)) ⟶ (∃AA∈ℳ→R. (∑[E;{AAA,BB}])=(∑[BBE;{AA,id(ℳ)}])
∧ (∀x∈ℳ-BBE. AAx=0)) by auto
    }
    then have ∃d∈E. (( ∃AA∈ℳ→R. ∑[E-{d};{AAA,BB}]=∑[BB(E-{d});{AA,id(ℳ)}]
∧ (∀x∈ℳ-BB(E-{d}). AAx=0)) ⟶ (∃AA∈ℳ→R. ∑[E;{AAA,BB}]=∑[BB(E);{AA,id(ℳ)}]
∧ (∀x∈ℳ-BBE. AAx=0)))
      using E(2) by auto
  }
  then have ∀E∈FinPow(X). E≠0 ⟶ (∃d∈E. ((∃AA∈ℳ→R. ∑[E-{d};{AAA,BB}]=∑[BB(E-{d});{A
(∀x∈ℳ-BB(E-{d}). AAx=0)) ⟶ (∃AA∈ℳ→R. ∑[E;{AAA,BB}]=∑[BB(E);{AA,id(ℳ)}]∧
```

```
(∀x∈ℳ-BBE. AAx=0))))
    by auto
  then show thesis using FinPow_ind_rem_one[where P=λE. ∃AA∈ℳ→R.
∑[E;{AAA,BB}]=∑[BBE;{AA,id(ℳ)}]∧ (∀x∈ℳ-BBE. AAx=0), OF case0 _ assms(3)]
by auto
qed
```

A span over a set is the collection over all linear combinations on those elements.

```
definition(in module0)
  Span({span of}_)
  where T⊆ℳ ⟹ {span of}T ≡ if T=0 then {Θ} else {∑[F;{AA,id(T)}].
⟨F,AA⟩∈{⟨FF,B⟩∈FinPow(T)×(T→R). ∀m∈T-FF. Bm=0}}
```

The span of a subset is then a submodule and contains the original set.

```
theorem(in module0) linear_ind_set_comb_submodule:
  assumes T⊆ℳ
  shows IsAsubmodule({span of}T)
  and T⊆{span of}T
proof-
  have id(T):T→T unfolding id_def by auto
  then have idG:id(T):T→ℳ using assms func1_1_L1B by auto
  {
    assume A:T=0
    from A assms have eq:({span of}T)={Θ} using Span_def by auto
    have ∀r∈R. r·SΘ=Θ using zero_fixed by auto
    with eq have ∀r∈R. ∀g∈({span of}T). r·Sg∈({span of}T) by auto more-
over
    from eq have IsAsubgroup(({span of}T),A_M) using mod_ab_gr.trivial_normal_subgroup
      unfolding IsAnormalSubgroup_def by auto
    ultimately have IsAsubmodule({span of}T) unfolding IsAsubmodule_def
by auto
    with A have IsAsubmodule({span of}T) T⊆{span of}T by auto
  }
  moreover
  {
    assume A:T≠0
    {
      fix t assume asT:t∈T
      then have Finite({t}) by auto
      with asT have FP:{t}∈FinPow(T) unfolding FinPow_def by auto more-
over
      from asT have Af:{⟨t,1⟩}∪{⟨r,0⟩. r∈T-{t}}:T→R unfolding Pi_def
function_def using
        Ring_ZF_1_L2(1,2) by auto moreover
      {
        fix m assume m∈T-{t}
        with Af have ({⟨t,1⟩}∪{⟨r,0⟩. r∈T-{t}})m=0 using apply_equality
by auto
```

}
ultimately have ∑ [{t};{{⟨t,**1**⟩}∪{⟨r,**0**⟩. r∈T-{t}},id(T)}]∈{span of}T
unfolding Span_def[OF assms(1)] **using** A
by auto
then have (({⟨t,**1**⟩}∪{⟨r,**0**⟩. r∈T-{t}})t) $\cdot_S$ (id(T)t) ∈{span of}T
using linComb_one_element[OF asT
Af idG] **by** auto
then have (({⟨t,**1**⟩}∪{⟨r,**0**⟩. r∈T-{t}})t) $\cdot_S$ t ∈{span of}T **using** asT
by auto
then have (**1**) $\cdot_S$ t ∈{span of}T **using** apply_equality Af **by** auto
moreover
have (**1**) $\cdot_S$ t=t **using** module_ax4 assms asT **by** auto **ultimately**
have t∈{span of}T **by** auto
}
then have T⊆{span of}T **by** auto **moreover**
with A **have** ({span of}T)≠0 **by** auto **moreover**
{
fix l **assume** l∈{span of}T
with A **obtain** S AA **where** l:l=∑ [S;{AA,id(T)}] S∈FinPow(T) AA:T→R
∀m∈T-S. AAm=**0** **unfolding** Span_def[OF assms(1)]
by auto
from l(1) **have** l∈$\mathcal{M}$ **using** linComb_is_in_module[OF l(3) _ l(2),
of id(T)] idG **unfolding** FinPow_def **by** auto
}
then have sub:({span of}T)⊆$\mathcal{M}$ **by** force **moreover**
{
fix T1 T2 **assume** as:T1∈{span of}T
T2∈{span of}T
with A **obtain** TT1 AA1 TT2 AA2 **where** T:TT1∈FinPow(T) TT2∈FinPow(T)
AA1:T→R
AA2:T→R T1=∑ [TT1;{AA1,id(T)}] T2=∑ [TT2;{AA2,id(T)}] ∀m∈T-TT1.
AA1m=**0** ∀m∈T-TT2. AA2m=**0** **unfolding** Span_def[OF assms(1)] **by** auto
{
assume A:TT1=0
then have T1=∑ [0;{AA1,id(T)}] **using** T(5) **by** auto
also have ...=Θ **using** LinearComb_def[OF T(3) idG T(1)] A **by** auto
ultimately have T1+$_V$T2=Θ+$_V$T2 **by** auto
also have ...∈{span of}T **using** sub as(2) mod_ab_gr.group0_2_L2
by auto
ultimately have T1+$_V$T2∈{span of}T **by** auto
}
moreover
{
assume AA:TT1≠0
have T1+$_V$T2=∑ [TT1+TT2;{{⟨⟨0,x⟩,AA1x⟩. x∈T}∪{⟨⟨1,x⟩,AA2x⟩. x∈T},{⟨⟨0,x⟩,id(T)x⟩.
x∈T}∪{⟨⟨1,x⟩,id(T)x⟩. x∈T}}] **using** linComb_sum[OF T(3,4) idG idG
AA T(1,2)] T(5,6) **by** auto
also have ...=∑ [TT1+TT2;{{⟨⟨0,x⟩,AA1x⟩. x∈T}∪{⟨⟨1,x⟩,AA2x⟩. x∈T},{⟨⟨0,x⟩,x⟩.
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T}}] **by** auto

```
        ultimately have eq:T1+_V T2=∑[TT1+TT2;{{⟨⟨0,x⟩,AA1x⟩. x∈T}∪{⟨⟨1,x⟩,AA2x⟩.
x∈T},{⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T}}] by auto
        from T(1,2) obtain n1 n2 where fin:TT1≈n1 n1∈nat TT2≈n2 n2∈nat
unfolding FinPow_def Finite_def by auto
        then have n1+n2≈n1#+n2 using nat_sum_eqpoll_sum by auto more-
over
        with fin(1,3) have TT1+TT2≈n1#+n2 using sum_eqpoll_cong[] eqpoll_trans[of
TT1+TT2 n1+n2 n1#+n2] by auto
        then have Finite(TT1+TT2) unfolding Finite_def using add_type
by auto
        then have fin:TT1+TT2∈FinPow(T+T) using T(1,2) unfolding FinPow_def
by auto moreover
        have f1:{⟨⟨0, x⟩, AA1  x⟩ . x ∈ T}:{0}×T→R using apply_type[OF
T(3)] unfolding Pi_def function_def by auto
        have f2:{⟨⟨1, x⟩, AA2  x⟩ . x ∈ T}:{1}×T→R using apply_type[OF
T(4)] unfolding Pi_def function_def by auto
        have ({0}×T)∩({1}×T)=0 by auto
        then have ffA:({⟨⟨0, x⟩, AA1  x⟩ . x ∈ T} ∪ {⟨⟨1, x⟩, AA2  x⟩ .
x ∈ T}):T+T→R unfolding sum_def using fun_disjoint_Un[OF f1 f2] by auto
moreover
        have f1:{⟨⟨0, x⟩, x⟩ . x ∈ T}:{0}×T→ℳ using assms unfolding
Pi_def function_def by blast
        have f2:{⟨⟨1, x⟩, x⟩ . x ∈ T}:{1}×T→ℳ using assms unfolding
Pi_def function_def by blast
        have f1T:{⟨⟨0, x⟩, x⟩ . x ∈ T}:{0}×T→T using assms unfolding
Pi_def function_def by blast
        have f2T:{⟨⟨1, x⟩, x⟩ . x ∈ T}:{1}×T→T using assms unfolding
Pi_def function_def by blast
        have ({0}×T)∩({1}×T)=0 by auto
        then have ffB:({⟨⟨0, x⟩, x⟩ . x ∈ T} ∪ {⟨⟨1, x⟩, x⟩ . x ∈T}):T+T→ℳ
and ffBT:({⟨⟨0, x⟩, x⟩ . x ∈ T} ∪ {⟨⟨1, x⟩, x⟩ . x ∈T}):T+T→T unfold-
ing sum_def using fun_disjoint_Un[OF f1 f2]
          using fun_disjoint_Un[OF f1T f2T] by auto
        obtain AA where AA:∑[TT1+TT2;{{⟨⟨0,x⟩,AA1x⟩. x∈T}∪{⟨⟨1,x⟩,AA2x⟩.
x∈T},{⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T}}]=
          ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{AA,id(ℳ)}]
AA:ℳ→R ∀x∈ℳ-({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2). AAx=0
          using index_module[OF ffA ffB fin] by auto
        from ffBT have sub:({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)⊆T
using func1_1_L6(2) by auto
        then have finpow:({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)∈FinPow(T)
using fin Finite_RepFun[of TT1+TT2 λt. ({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})t]

          func_imagedef[OF ffBT, of TT1+TT2] unfolding FinPow_def by auto
        {
          fix R T assume R∈Pow(T)
          then have id(R)⊆R*T using func1_1_L1B[OF id_type] by auto
          then have id(R)=restrict(id(T),R) using right_comp_id_any[of
id(T)] using left_comp_id[of id(R) R T] by force
```

```
        }
        then have reg:∀T. ∀R∈Pow(T). id(R)=restrict(id(T),R) by blast
        then have ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T)}]=
          ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(restrict(AA,T),({⟨⟨0,x⟩,x
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)),id(({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2))}]
          using linComb_restrict_coord[OF restrict_fun[OF AA(2) assms]
func1_1_L1B[OF id_type assms] finpow] sub by auto moreover
        from sub have  T ∩ ({⟨⟨0, x⟩, x⟩ . x ∈ T} ∪ {⟨⟨1, x⟩, x⟩ . x ∈
T})  (TT1 + TT2)=({⟨⟨0, x⟩, x⟩ . x ∈ T} ∪ {⟨⟨1, x⟩, x⟩ . x ∈ T})  (TT1
+ TT2) by auto
        then have restrict(restrict(AA,T),({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩.
x∈T})(TT1+TT2))=restrict(AA,({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2))
          using restrict_restrict[of AA T ({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩.
x∈T})(TT1+TT2)] by auto
        ultimately have ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T
          ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,({⟨⟨0,x⟩,x⟩.
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)),id(({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2))}]
by auto
        moreover have ({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)⊆𝓜
using sub assms by auto
        with reg have id(({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2))=restrict(id(𝓜),({⟨⟨0
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)) by auto
        ultimately have ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T
          ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,({⟨⟨0,x⟩,x⟩.
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)),restrict(id(𝓜),({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩.
x∈T})(TT1+TT2))}]
          by auto
        also have ...=∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{AA,id(𝓜)}]
using linComb_restrict_coord[OF AA(2) id_type, of ({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩.
x∈T})(TT1+TT2)]
          finpow unfolding FinPow_def using assms by force
        ultimately have eq1:∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),
          ∑[({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{AA,id(𝓜)}]
by auto
        have restrict(AA,T):T→R using restrict_fun[OF AA(2) assms]. more-
over
        note finpow moreover
        {
          fix x assume x:x∈T-({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)
          with assms have x∈𝓜-({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)
by auto
          with AA(3) have AAx=0 by auto
          with x have restrict(AA,T)x=0 using restrict by auto
        }
        then have ∀x∈T-({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2). restrict(AA,T)x=0
by auto ultimately
        have ⟨({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2),restrict(AA,T)⟩∈{⟨F,E⟩∈FinPow(T)×
∀x∈T-F. Ex=0} by auto
        then have ∃F∈FinPow(T). ∃E∈T→R. (∀x∈T-F. Ex=0) ∧ (∑[({⟨⟨0,x⟩,x⟩.
```

```
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T)}]=∑ [F;{E,id(T)}])
          using exI[of λF. F∈FinPow(T) ∧ (∃E∈T→R. (∀x∈T-F. Ex=0) ∧
(∑ [({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T)}]=∑ [F;{E,id(T)}]))
({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)]
          exI[of λE. E∈T→R ∧ ((∀x∈T-(({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2)).
Ex=0) ∧ (∑ [({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T)}]=∑ [({⟨⟨0,x⟩,
x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{E,id(T)}]))
            restrict(AA,T)] by auto
        then have ∑ [({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{restrict(AA,T),id(T)}]∈{

          ⟨FF,AA⟩ ∈ {⟨F,E⟩∈FinPow(T)×(T→R). ∀x∈T-F. Ex=0}} by auto
        with eq1 have ∑ [({⟨⟨0,x⟩,x⟩. x∈T}∪{⟨⟨1,x⟩,x⟩. x∈T})(TT1+TT2);{AA,id(𝓜)}]∈{span
of}T using A unfolding Span_def[OF assms]
          by auto
        with AA(1) eq have T1+_V T2∈{span of}T by auto
      }
      ultimately have T1+_V T2∈{span of}T by auto
    }
    then have ∀x∈{span of}T. ∀y∈{span of}T. x+_V y∈{span of}T by blast
    then have ({span of}T){is closed under}A_M  unfolding IsOpClosed_def
by auto moreover
    {
      fix r TT assume r∈R TT∈{span of}T
      with A obtain n AA where T:r∈R TT=∑ [n;{AA,id(T)}] n∈FinPow(T)
AA:T→R ∀x∈T-n. AAx=0
        unfolding Span_def[OF assms(1)] by auto
      have r·_S(TT)=∑ [n;{{⟨t,r·(AAt)⟩.t∈T},id(T)}] using linComb_action(1)[OF
T(4) _ T(1) T(3), of id(T)] T(2)
        func1_1_L1B[OF id_type assms] by auto moreover
      have fun:{⟨t,r·(AAt)⟩.t∈T}:T→R using linComb_action(2)[OF T(4) _
T(1,3)] func1_1_L1B[OF id_type assms] by auto
        moreover
      {
        fix z assume z:z∈T-n
        then have {⟨t,r·(AAt)⟩.t∈T}z=r·(AAz) using apply_equality[of z
r·(AAz)
          {⟨t,r·(AAt)⟩.t∈T} T λt. R] using fun by auto
        also have ...=r·0 using T(5) z by auto
        also have ...=0 using Ring_ZF_1_L6(2)[OF T(1)] by auto
        ultimately have {⟨t,r·(AAt)⟩.t∈T}z=0 by auto
      }
      then have ∀z∈T-n. {⟨t,r·(AAt)⟩.t∈T}z=0 by auto ultimately
      have r·_S(TT)∈{span of}T unfolding Span_def[OF assms(1)] using A
T(3) by auto
    }
    then have ∀r∈R. ∀TT∈{span of}T. r·_S(TT)∈{span of}T by blast
    ultimately have IsAsubmodule({span of}T) T⊆({span of}T) using submoduleI
by auto
  }
```

```
  ultimately show IsAsubmodule({span of}T) T⊆({span of}T) by auto
qed
```

Given a linear combination, it is in the span of the image of the second function.

```
lemma (in module0) linear_comb_span:
  assumes AA:X→R B:X→ℳ D∈FinPow(X)
  shows ∑[D;{AA,B}]∈({span of}(BD))
proof-
  {
    assume A:BD=0
    {
      assume D≠0
      then obtain d where d∈D by auto
      then have Bd∈BD using image_fun[OF assms(2)] assms(3) unfolding
FinPow_def by auto
      with A have False by auto
    }
    then have D=0 by auto
    then have ∑[D;{AA,B}]=Θ using LinearComb_def[OF assms] by auto
moreover
    with A have thesis using Span_def by auto
  }
  moreover
  {
    assume A:BD≠0
    have sub:BD⊆ℳ using assms(2) func1_1_L6(2) by auto
    from assms obtain AB where AA:∑[D;{AA,B}]=∑[BD;{AB,id(ℳ)}] AB:ℳ→R
∀x∈ℳ-BD. ABx=0
      using index_module by blast
    have fin:Finite(BD) using func_imagedef[OF assms(2)] assms(3) un-
folding FinPow_def
      using Finite_RepFun[of D λt. Bt] by auto
    with sub have finpow:BD∈FinPow(ℳ) unfolding FinPow_def by auto
    with AA(1) have ∑[D;{AA,B}]=∑[BD;{restrict(AB,BD),restrict(id(ℳ),BD)}]
      using linComb_restrict_coord AA(2) id_type by auto
    moreover have restrict(id(ℳ),BD)=id(ℳ) O id(BD) using right_comp_id_any
by auto
    then have restrict(id(ℳ),BD)=id(BD) using left_comp_id[of id(BD)]
sub by auto
    ultimately have ∑[D;{AA,B}]=∑[BD;{restrict(AB,BD),id(BD)}] by auto
moreover
    have restrict(AB,BD):BD→R using AA(2) restrict_fun sub by auto more-
over
    have ∀x∈BD-BD. restrict(AB,BD)x=0_R by auto ultimately
    have ∑[D;{AA,B}]∈{span of}(BD) using fin A unfolding Span_def[OF
sub] FinPow_def by auto
  }
  ultimately show thesis by blast
```

qed

It turns out that the span is the smallest submodule that contains the original set.

```
theorem(in module0) minimal_submodule:
  assumes T⊆𝒩 IsAsubmodule(𝒩)
  shows ({span of}T)⊆𝒩
proof-
  have as:T⊆ℳ using assms(1) mod_ab_gr.group0_3_L2[OF sumodule_is_subgroup[OF
assms(2)]] by auto
  {
    assume A:T=0
    {
      fix x assume x∈{span of}T
      with A have x=Θ using Span_def[OF as] by auto
      then have x∈𝒩 using assms(2) unfolding IsAsubmodule_def
        using mod_ab_gr.group0_3_L5 by auto
    }
    then have ({span of}T)⊆𝒩 by auto
  }
  moreover
  {
    assume A:T≠0
    {
      fix x assume x∈{span of}T
      with A obtain n AA where x=∑[n;{AA,id(T)}] n∈FinPow(T) AA:T→R
        unfolding Span_def[OF as] by auto
      then have x:x=∑[n;{AA,id(T)}] n∈FinPow(T) AA:T→R id(T):T→𝒩
        using assms(1) func1_1_L1B id_type[of T] by auto
      then have x∈𝒩 using linear_comb_submod assms(2) by auto
    }
    then have ({span of}T)⊆𝒩 by auto
  }
  ultimately show ({span of}T)⊆𝒩 by auto
qed

end
```

```
theory Module_ZF_2 imports Module_ZF_1 Ring_ZF_2

begin
```

The most basic examples of modules, are subsets of the ring; since a ring is an abelian group when considering addition.

52.5 Ideals as Modules

Let's show first that the ring acting on itself is a module; and then we will show that ideals are submodules.

The map that takes every element to its left multiplication map, is a map to endomorphisms.

```
lemma (in ring0) action_regular_map:
  shows {⟨r,{⟨s,r·s⟩. s∈R}⟩. r∈R}:R→End(R,A) unfolding Pi_def
    function_def End_def Homomor_def IsMorphism_def apply auto
  using Ring_ZF_1_L4(3) apply simp using Ring_ZF_1_L4(3) apply simp
proof-
  fix x g1 g2 assume as:x∈R g1∈R g2∈R
  have f:{⟨t, M ⟨x, t⟩⟩ . t ∈ R}:R→R unfolding Pi_def function_def ap-
ply auto
    using Ring_ZF_1_L4(3) as(1) by auto
  from f as(2) have g1:{⟨t, M ⟨x, t⟩⟩ . t ∈ R}g1 = x·g1 using apply_equality
by auto
  from f as(3) have g2:{⟨t, M ⟨x, t⟩⟩ . t ∈ R}g2 = x·g2 using apply_equality
by auto
  have g1+g2 ∈R using Ring_ZF_1_L4(1) as(3,2) by auto
  then have {⟨t, M ⟨x, t⟩⟩ . t ∈ R} (g1+g2) = x·(g1+g2) using apply_equality
f by auto
  with g1 g2 show {⟨t, M ⟨x, t⟩⟩ . t ∈ R} (A ⟨g1, g2⟩) = A ⟨{⟨t, M
⟨x, t⟩⟩ . t ∈ R} g1, {⟨t, M ⟨x, t⟩⟩ . t ∈ R} g2⟩
    using ring_oper_distr(1)[of x g1 g2] as by auto
qed
```

The previous map respects addition because of distribution

```
lemma (in ring0) action_regular_distrib:
  assumes g₁ ∈ R g₂ ∈ R
  shows {⟨xa, M ⟨(A ⟨g₁, g₂⟩), xa⟩⟩ . xa ∈ R} =
      EndAdd(R, A) ⟨{⟨xa, M ⟨g₁, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g₂, xa⟩⟩ .
xa ∈ R} ⟩
proof(rule func_eq)
  from assms have A:g₁+g₂∈R using Ring_ZF_1_L4(1) by auto
  then have {⟨r,{⟨s,r·s⟩. s∈R}⟩. r∈R}(g₁+g₂) = {⟨xa, M ⟨(A ⟨g₁, g₂⟩),
xa⟩⟩ . xa ∈ R} using
    apply_equality action_regular_map by auto
  then show f1: {⟨xa, M ⟨(A ⟨g₁, g₂⟩), xa⟩⟩ . xa ∈ R}:R→R using
    apply_type[OF action_regular_map A] unfolding End_def by auto
  have END:{⟨xa, M ⟨g₁, xa⟩⟩ . xa ∈ R}:End(R,A) {⟨xa, M ⟨g₂, xa⟩⟩ . xa
∈ R}:End(R,A) using
     apply_type[OF action_regular_map assms(1)] apply_type[OF action_regular_map
assms(2)]
    apply_equality[OF _ action_regular_map] assms by auto
  with apply_type[OF restrict_fun[OF monoid0.Group_ZF_2_1_L0A, of R A
_ R End(R,A)×End(R,A)]]
  show f2:EndAdd(R, A)
```

⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩:R→R

unfolding EndAdd_def End_def **using** Ring_ZF_1_L1(2) **unfolding** group0_def monoid0_def IsAgroup_def **by** blast
{
fix x **assume** x:x∈R
then have {⟨xa, M ⟨A ⟨g_1, g_2⟩, xa⟩⟩ . xa ∈ R} x = (g_1+g_2)·x **using** apply_equality[OF _ f1] **by** auto
moreover
from END **have** EndAdd(R, A) ⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩ x = (A {lifted to function space over} R)⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩ x **using** restrict
unfolding EndAdd_def **by** auto
with END **have** EndAdd(R, A) ⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩ x = A⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}x, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}x⟩
using group0.Group_ZF_2_1_L3[OF Ring_ZF_1_L1(2) _ _ _ x] **unfolding** EndAdd_def End_def **by** auto
then have EndAdd(R, A) ⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩ x = (g_1·x)+(g_2·x) **using** apply_equality x
END **unfolding** End_def **by** auto
moreover
have (g_1·x)+(g_2·x)= (g_1+g_2)·x **using** ring_oper_distr(2) x assms **by** auto
ultimately have {⟨xa, M ⟨A ⟨g_1, g_2⟩, xa⟩⟩ . xa ∈ R} x = EndAdd(R, A) ⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩ x **by** auto
}
then show ∀x∈R. {⟨xa, M ⟨A ⟨g_1, g_2⟩, xa⟩⟩ . xa ∈ R} x = EndAdd(R, A) ⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}⟩ x **by** auto
qed

The previous map respects multiplication because of associativity

lemma (**in** ring0) action_regular_assoc:
assumes g_1 ∈ R g_2 ∈ R
shows {⟨xa, M ⟨(M ⟨g_1, g_2⟩), xa⟩⟩ . xa ∈ R} =
EndMult(R, A) ⟨{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R} ⟩
proof(rule func_eq)
from assms **have** A:g_1·g_2∈R **using** Ring_ZF_1_L4(3) **by** auto
then have {⟨r,{⟨s,r·s⟩. s∈R}⟩. r∈R}(g_1·g_2) = {⟨xa, M ⟨(M ⟨g_1, g_2⟩), xa⟩⟩ . xa ∈ R} **using**
apply_equality action_regular_map **by** auto
then show f1: {⟨xa, M ⟨(M ⟨g_1, g_2⟩), xa⟩⟩ . xa ∈ R}:R→R **using**
apply_type[OF action_regular_map A] **unfolding** End_def **by** auto
have END:{⟨xa, M ⟨g_1, xa⟩⟩ . xa ∈ R}:End(R,A) {⟨xa, M ⟨g_2, xa⟩⟩ . xa ∈ R}:End(R,A) **using**
apply_type[OF action_regular_map assms(1)] apply_type[OF action_regular_map assms(2)]
apply_equality[OF _ action_regular_map] assms **by** auto
with apply_type[OF restrict_fun[OF func_ZF_5_L1[of R], of End(R,A)×End(R,A)]]

show f2:EndMult(R, A)
$\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$:R$\rightarrow$R

unfolding EndMult_def End_def **unfolding** group0_def monoid0_def IsAgroup_def **by** blast
{
fix x **assume** x:x$\in$R
then have \{$\langle$xa, M $\langle$M $\langle g_1$, $g_2\rangle$, xa$\rangle\rangle$. xa $\in$ R\} x = $(g_1 \cdot g_2) \cdot$x **using** apply_equality[OF _ f1] **by** auto
moreover
from END **have** EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x = Composition(R)$\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x **using** restrict
unfolding EndMult_def **by** auto
with END **have** EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x = (\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\} O \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\})x
unfolding End_def **using** func_ZF_5_L2 **by** auto
then have EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x = \{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\} (\{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}x) **using** x
END comp_fun_apply[of \{$\langle$t,$g_2\cdot$t$\rangle$. t$\in$R\} R R x] **unfolding** End_def **by** auto
then have EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x = \{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\} $(g_2\cdot$x)
using apply_equality END(2) x **unfolding** End_def **by** auto
then have EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x = $g_1\cdot(g_2\cdot$x)
using apply_equality END(1) Ring_ZF_1_L4(3)[OF assms(2) x] **unfolding** End_def **by** auto
moreover
have $g_1\cdot g_2\cdot$x= $g_1\cdot(g_2\cdot$x) **using** Ring_ZF_1_L11(2) x assms **by** auto
ultimately have \{$\langle$xa, M $\langle$M $\langle g_1$, $g_2\rangle$, xa$\rangle\rangle$. xa $\in$ R\} x = EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x **by** auto
}
then show $\forall$x$\in$R. \{$\langle$xa, M $\langle$M $\langle g_1$, $g_2\rangle$, xa$\rangle\rangle$. xa $\in$ R\} x = EndMult(R, A) $\langle$\{$\langle$xa, M $\langle g_1$, xa$\rangle\rangle$. xa $\in$ R\}, \{$\langle$xa, M $\langle g_2$, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$ x **by** auto
qed

The previous map takes the unit element to the identity map

lemma (**in** ring0) action_regular_neut:
shows \{$\langle$x, \{$\langle$xa, M $\langle$x, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$. x $\in$ R\} **1** = id(R)
proof (rule func_eq)
from apply_type[OF action_regular_map] **have** \{$\langle$x, \{$\langle$xa, M $\langle$x, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$. x $\in$ R\} **1** :End(R,A)
using Ring_ZF_1_L2(2) **by** auto
then show f1:\{$\langle$x, \{$\langle$xa, M $\langle$x, xa$\rangle\rangle$. xa $\in$ R\}$\rangle$. x $\in$ R\} **1** :R$\rightarrow$R **unfolding** End_def **by** auto

```
  show id(R):R→R using id_inj unfolding inj_def by auto
  {
    fix x assume x:x∈R
    have ({⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  1) = {⟨xa, M  ⟨1,
xa⟩⟩ . xa ∈ R}
      using apply_equality[OF _ action_regular_map] Ring_ZF_1_L2(2) by
auto
    with x f1 have ({⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  1)x =
1·x using apply_equality
      by auto
    then have ({⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  1)x = x us-
ing Ring_ZF_1_L3(6) x by auto
    then have ({⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  1)x = id(R)x
using x by auto
  }
  then show  ∀x∈R. {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  1  x =
id(R)  x by auto
qed
```

The previous map is an action

```
theorem(in ring0) action_regular:
  shows IsAction(R,A,M,R,A,{⟨r,{⟨s,r·s⟩. s∈R}⟩. r∈R}) unfolding IsAction_def
ringHomomor_def
  IsMorphism_def apply auto using action_regular_map apply simp prefer
3
  using group0.end_comp_monoid(2)[OF Ring_ZF_1_L1(2)] action_regular_neut
unfolding EndMult_def apply simp
proof-
  fix g₁ g₂ assume as: g₁ ∈ R g₂ ∈ R
  have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  (A  ⟨g₁, g₂⟩) = {⟨xa,
M  ⟨A  ⟨g₁, g₂⟩, xa⟩⟩ . xa ∈ R}
    using apply_equality[OF _ action_regular_map] Ring_ZF_1_L4(1) as by
auto moreover
  have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g₁ = {⟨xa, M  ⟨g₁, xa⟩⟩
. xa ∈ R}
    using apply_equality[OF _ action_regular_map] as(1) by auto more-
over
  have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g₂ = {⟨xa, M  ⟨g₂, xa⟩⟩
. xa ∈ R}
    using apply_equality[OF _ action_regular_map] as(2) by auto more-
over
  note action_regular_distrib[OF as] ultimately
  show {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  (A  ⟨g₁, g₂⟩) =
       EndAdd(R, A)  ⟨{⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g₁, {⟨x,
{⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g₂⟩ by auto
  have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  (M  ⟨g₁, g₂⟩) = {⟨xa,
M  ⟨M  ⟨g₁, g₂⟩, xa⟩⟩ . xa ∈ R}
    using apply_equality[OF _ action_regular_map] Ring_ZF_1_L4(3) as by
auto moreover
```

```
  have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g1 = {⟨xa, M  ⟨g1, xa⟩⟩
. xa ∈ R}
    using apply_equality[OF _ action_regular_map] as(1) by auto more-
over
  have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g2 = {⟨xa, M  ⟨g2, xa⟩⟩
. xa ∈ R}
    using apply_equality[OF _ action_regular_map] as(2) by auto more-
over
  note action_regular_assoc[OF as] ultimately
  show {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  (M  ⟨g1, g2⟩) =
        (Composition(R) {in End} [R,A])  ⟨{⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈
R}⟩ . x ∈ R}  g1, {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  g2⟩
    unfolding EndMult_def by auto
qed
```

The action defines the Regular Module

```
theorem (in ring0) reg_module:
  shows module0(R,A,M,R,A,{⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}) un-
folding module0_def
  module0_axioms_def using ring0_axioms action_regular unfolding ring0_def
IsAring_def by auto
```

Every ideal is a submodule of this regular action.

```
corollary (in ring0) ideal_submodule:
  assumes I◁R
  shows module0.IsAsubmodule(R,A,{⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x
∈ R},I)
proof(rule module0.submoduleI[OF reg_module])
  from ideal_dest_subset[OF assms] show I ⊆ R by auto
  from ideal_dest_zero[OF assms] show I≠0 by auto
  {
    fix r h assume as:r:R h:I
    then have A:{⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  r = {⟨xa, M
⟨r, xa⟩⟩ . xa ∈ R}
      using apply_equality[OF _ action_regular_map] by auto
    then have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  r h= {⟨xa, M
⟨r, xa⟩⟩ . xa ∈ R}h by auto
    moreover from A have {⟨xa, M  ⟨r, xa⟩⟩ . xa ∈ R}:R→R using apply_type[OF
action_regular_map as(1)]
      unfolding End_def by auto
    then have {⟨xa, M  ⟨r, xa⟩⟩ . xa ∈ R}h = r·h using apply_equality as(2)
ideal_dest_subset[OF assms] by auto
    ultimately have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  r h=r·h
by auto
    then have {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  r h:I using
as ideal_dest_mult(2) assms by auto
  }
  then show ∀r∈R. ∀h∈I. {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  r
h ∈ I by auto
```

```
  show I{is closed under}A using assms ideal_dest_sum unfolding IsOpClosed_def
by auto
qed
```

52.6 Annihilators

An annihilator of a module subset is the set of elements of the ring whose action on that module subset is 0.

```
definition (in module0) ann where
N ⊆ 𝓜 ⟹ ann(N) ≡ {r∈R. ∀n∈N. r·_S n =Θ}
```

If the subset is a submodule, then the annihilator is an ideal.

```
lemma (in module0) ann_ideal:
  assumes IsAsubmodule(N)
  shows ann(N)◁R unfolding Ideal_def
proof(safe)
  have sub:N ⊆ 𝓜 using mod_ab_gr.group0_3_L2 using assms unfolding IsAsubmodule_def
by auto
  fix x y assume as:x∈ann(N) y∈R
  {
    fix n assume n:n∈N
    then have (x·y)·_S n=x·_S(y·_S n) using module_ax3 as assms sub unfold-
ing ann_def[OF sub]
      by auto
    moreover have y·_S n∈N using n as(2) sumodule_is_subaction[OF assms]
by auto
    with as(1) have x·_S(y·_S n) = Θ unfolding ann_def[OF sub] by auto
    ultimately have (x·y)·_S n=Θ by auto
  }
  then have ∀n∈N. (x·y)·_S n=Θ by auto
  then show x·y∈ann(N) using as Ring_ZF_1_L4(3) unfolding ann_def[OF
sub] by auto
  {
    fix n assume n:n∈N
    then have (y·x)·_S n=y·_S(x·_S n) using module_ax3 as assms sub unfold-
ing ann_def[OF sub] by auto
    moreover have x·_S n=Θ using n as(1) unfolding ann_def[OF sub] by auto
    with as(2) have y·_S(x·_S n) = Θ using zero_fixed by auto
    ultimately have (y·x)·_S n=Θ by auto
  }
  then have ∀n∈N. (y·x)·_S n=Θ by auto
  then show y·x∈ann(N) using as Ring_ZF_1_L4(3) unfolding ann_def[OF
sub] by auto
next
  have sub:N ⊆ 𝓜 using mod_ab_gr.group0_3_L2 using assms unfolding IsAsubmodule_def
by auto
  show IsAsubgroup(ann(N),A)
  proof(rule add_group.group0_3_T3)
```

```
    show ann(N) ⊆ R unfolding ann_def[OF sub] by auto
    have 0∈R using Ring_ZF_1_L2(1) by auto
    moreover have ∀n∈N. 0·Sn=Θ using mult_zero sub by auto
    ultimately have 0∈ann(N) unfolding ann_def[OF sub] by auto
    then show ann(N)≠0 by auto
    {
      fix x assume x∈ann(N)
      then have x:x∈R ∀n∈N. x·Sn=Θ unfolding ann_def[OF sub] by auto
      {
        fix n assume n:n∈N
        have (-x)·Sn=(x·(-1))·Sn using Ring_ZF_1_L7(2)[of x 1] Ring_ZF_1_L2(2)
          Ring_ZF_1_L3(5)[of x] x(1) by auto
        then have (-x)·Sn=x·S((-1)·Sn) using module_ax3[of x -1 n]
          using n sub x(1) Ring_ZF_1_L3(1)[OF Ring_ZF_1_L2(2)] by auto
        moreover have (-1)·Sn∈N using sumodule_is_subaction[OF assms
Ring_ZF_1_L3(1)[OF Ring_ZF_1_L2(2)] n].
        then have x·S((-1)·Sn) = Θ using x(2) by auto
        ultimately have (-x)·Sn=Θ by auto
      }
      then have ∀n∈N. (-x)·Sn=Θ by auto moreover
      have (-x)∈R using Ring_ZF_1_L3(1) x(1) by auto
      ultimately have (-x) ∈ ann(N) unfolding ann_def[OF sub] by auto
    }
    then show ∀x∈ann(N). (-x) ∈ ann(N) by auto
    {
      fix x y assume as:x∈ann(N) y∈ann(N)
      from as(1) have x:x∈R unfolding ann_def[OF sub] by auto
      from as(2) have y:y∈R unfolding ann_def[OF sub] by auto
      {
        fix n assume n:n∈N
        from n as(1) have x0:x·Sn=Θ unfolding ann_def[OF sub] by auto
        from n as(2) have y0:y·Sn=Θ unfolding ann_def[OF sub] by auto
        have (x+y)·Sn = (x·Sn) +V(y·Sn) using module_ax2[of x y n] x y
n sub by auto
        with x0 y0 have (x+y)·Sn =Θ +VΘ by auto
        then have (x+y)·Sn =Θ using zero_neutral(1) zero_in_mod by auto
      }
      then have ∀n∈N. (x+y)·Sn =Θ by auto moreover
      have x+y:R using Ring_ZF_1_L4(1) x y by auto
      ultimately have x+y∈ann(N) unfolding ann_def[OF sub] by auto
    }
    then show ann(N) {is closed under} A unfolding IsOpClosed_def by
auto
  qed
qed
```

Annihilator is reverse monotonic

```
lemma (in module0) ann_mono:
  assumes N ⊆ M K ⊆ N
```

```
  shows ann(N) ⊆ ann(K)
proof
  fix x assume x∈ann(N)
  then have x∈R ∀n∈N. x·Sn =Θ unfolding ann_def[OF assms(1)] by auto
  then have x∈R ∀n∈K. x·Sn =Θ using assms(2) by auto
  then have x∈ {r ∈ R . ∀n∈K. r ·S n = Θ} by auto moreover
  from assms have K ⊆ ℳ by auto
  ultimately show x∈ann(K) using ann_def[of K] by auto
qed
```

If the ring is commutative, the annihilator of a subset shrinks to the annihilator of the generated submodule

```
lemma (in module0) comm_ann_of_ideal:
  assumes N ⊆ ℳ M {is commutative on} R
  shows ann(N) = ann({span of}N)
proof
  have N ⊆ {span of}N using linear_ind_set_comb_submodule(2) assms(1)
by auto
  moreover have ({span of}N) ⊆ ℳ using trivial_submodules(1)
    minimal_submodule[OF assms(1)] by auto moreover
  note assms(1) ultimately show ann({span of}N) ⊆ ann(N) using ann_mono
by auto
  {
    fix x assume x∈ann(N)
    then have x:x∈R ∀n∈N. x·Sn =Θ unfolding ann_def[OF assms(1)] by
auto
    let Q={m∈ℳ. x·Sm=Θ}
    have IsAsubmodule(Q)
    proof (rule submoduleI)
      show Q ⊆ ℳ by auto
      have Θ∈Q using zero_fixed x(1) zero_in_mod by auto
      then show Q≠0 by auto
      {
        fix t h assume as:t∈R h:Q
        from as(2) have h:h∈ℳ x·Sh=Θ by auto
        have x·S(t·Sh) = (x·t)·Sh using module_ax3 as(1) x(1) h(1) by auto
        then have x·S(t·Sh) = (t·x)·Sh using as(1) x(1) assms(2) unfold-
ing IsCommutative_def by auto
        then have x·S(t·Sh) = t·S(x·Sh) using module_ax3 as(1) x(1) h(1)
by auto
        with h(2) have x·S(t·Sh) = t·SΘ by auto
        then have x·S(t·Sh) = Θ using zero_fixed as(1) by auto
        moreover have t·Sh∈ℳ using as(1) h(1) apply_type[OF H_val_type(2)]
by auto
        ultimately have t·Sh∈Q by auto
      }
      then show  ∀r∈R. ∀h∈{m ∈ ℳ . x ·S m = Θ}. r ·S h ∈ {m ∈ ℳ .
x ·S m = Θ} by auto
      {
```

```
        fix u v assume u:Q v:Q
        then have uv:u∈ℳ v∈ℳ x ·S u = Θ x ·S v = Θ by auto
        have x·S(u+Vv) = x ·S u +V x ·S v using module_ax1 uv(1,2) x(1)
by auto
        with uv(3,4) have x·S(u+Vv) = Θ +V Θ by auto
        then have x·S(u+Vv) = Θ using zero_neutral(1) zero_in_mod by
auto
        moreover have u+Vv:ℳ using mod_ab_gr.group_op_closed uv(1,2)
by auto
        ultimately have u+Vv:Q by auto
      }
      then show Q{is closed under}AM unfolding IsOpClosed_def by auto
    qed
    moreover have N ⊆ Q using assms(1) x(2) by auto
    ultimately have ({span of}N) ⊆ Q using minimal_submodule by auto
    then have ann(Q) ⊆ ann({span of}N) using ann_mono[of Q {span of}N]
by auto
    moreover have ann(Q) = {r∈R. ∀n∈Q. r·Sn = Θ} using ann_def[of Q]
by auto
    then have ann(Q) = {r∈R. ∀n∈ℳ. x·Sn=Θ ⟶ r·Sn = Θ} using ann_def[of
Q] by auto
    with x(1) have x∈ann(Q) by auto
    ultimately have x∈ann({span of}N) by auto
  }
  then show ann(N) ⊆ ann({span of}N) by auto
qed
```

Annihilators on commutative rings are ideals

```
corollary (in module0) comm_ann_ideal:
  assumes N ⊆ ℳ M {is commutative on} R
  shows ann(N) ◁R using comm_ann_of_ideal[OF assms] linear_ind_set_comb_submodule(1)[OF
assms(1)]
    ann_ideal by auto

end
```

53 Vector spaces

```
theory VectorSpace_ZF imports Module_ZF

begin
```

Vector spaces have a long history of applications in mathematics and physics. To this collection of applications a new one has been added recently - Large Language Models. It turned out that representing words, phrases and documents as vectors in a high-dimensional vector space provides an effective way to capture semantic relationships and emulate contextual understanding. This theory has nothing to do with LLM's however - it just defines vector space as a mathematical structure as it has been understood from at least the beginning of the XXth century.

53.1 Definition and basic properties of vector spaces

The canonical example of a vector space is $\mathbb{R}^n$ - the set of n-tuples of real numbers. We can add them adding respective coordinates and scale them by multiplying all coordinates by the same number. In a more abstract approach we start with an abelian group (of vectors) and a field (of scalars) and define an operation of multiplying a vector by a scalar so that the distributive properties $x(v_1 + v_2) = sv_1 + sv_2$ and $(s_1 + s_2)v = s_1 v + s_2 v$ are satisfied for any scalars s, s_1, s_2 and vectors v, v_1, v_2.

A vector space is a field action on an abelian group. The notion of an action is defined in `Module_ZF` theory as a ring homomorphism valuesd in the ring of endomorphisms of some (abelian) group. In the definition S is a the ring carrier, A is the set representing the addition operation of the ring, M is the ring multiplication, V is the carrier of the abelian group, A_V represents the group operation, i.e. the vector addition and H is the ring homomorphism defining the action.

```
definition IsVectorSpace(S,A,M,V,A_V,H) ≡
  IsAfield(S,A,M) ∧ IsAgroup(V,A_V) ∧ (A_V {is commutative on} V) ∧ IsAction(S,A,M,V,A_V,H)
```

The next locale defines context (i.e. common assumptions and notation) when considering vector spaces. We reuse notation from the `field0` locale adding more similarly to the `module0` locale.

```
locale vector_space0 = field0 +
  fixes V A_V H

  assumes mAbGr: IsAgroup(V,A_V) ∧ (A_V {is commutative on} V)

  assumes mAction: IsAction(K,A,M,V,A_V,H)

  fixes zero_vec (Θ)
  defines zero_vec_def [simp]: Θ ≡ TheNeutralElement(V,A_V)

  fixes vAdd (infixl +_V 80)
  defines vAdd_def [simp]: v₁ +_V v₂ ≡ A_V⟨v₁,v₂⟩

  fixes scal (infix ·_S 90)
  defines scal_def [simp]: s ·_S v ≡ (H(s))(v)

  fixes negV (−_)
  defines negV_def [simp]: −v ≡ GroupInv(V,A_V)(v)

  fixes vSub (infix -_V 80)
```

```
  defines vSub_def [simp]: v₁ -V v₂ ≡ v₁ +V (−v₂)
```

We indeed talk about vector spaces in the vector_space0 context.

```
lemma (in vector_space0) V_vec_space: shows IsVectorSpace(K,A,M,V,AV,H)
  using mAbGr mAction Field_ZF_1_L1 unfolding IsVectorSpace_def by simp
```

If a quintuple of sets forms a vector space then the assumptions of the vector_spce0 hold for those sets.

```
lemma vec_spce_vec_spce_contxt: assumes IsVectorSpace(K,A,M,V,AV,H)
  shows vector_space0(K, A, M, V, AV, H)
proof
  from assms show
    IsAring(K, A, M) M {is commutative on} K
    TheNeutralElement(K, A) ≠ TheNeutralElement(K, M)
    ∀x∈K. x ≠ TheNeutralElement(K, A) ⟶ (∃y∈K. M⟨x, y⟩ = TheNeutralElement(K,
M))
    IsAgroup(V, AV) ∧ AV {is commutative on} V
    IsAction(K, A, M, V, AV, H)
    unfolding IsAfield_def IsVectorSpace_def by simp_all
qed
```

The assumptions of module0 context hold in the vector_spce0 context.

```
lemma (in vector_space0) vec_spce_mod: shows module0(K, A, M, V, AV,
H)
proof
  from mAbGr show IsAgroup(V,AV) ∧ (AV {is commutative on} V) by simp
  from mAction show IsAction(K,A,M,V,AV,H) by simp
qed
```

Propositions proven in the module0 context are valid in the vector_spce0 context.

```
sublocale vector_space0 < vspce_mod: module0 K A M
    ringa ringminus ringsub ringm ringzero ringone ringtwo ringsq
    λs. Fold(A,0,s)
    λ n x. Fold(A,0,{⟨k,x⟩. k∈n})
    V AV
  using vec_spce_mod by auto
```

53.2 Vector space axioms

In this section we show that the definition of a vector space as a field action on an abelian group implies the vector space axioms as listed on Wikipedia (March 2024). The first four axioms just state that vectors with addition form an abelian group. That is fine of course, but in such case the axioms for scalars being a field should be listed too, and they are not. The entry on modules is more consistent, it states that module elements form an abelian group, scalars form a ring and lists only four properties of multiplication of scalars by vectors as module axioms. The remaining four axioms are just restatemenst of module axioms and since vector spaces are modules we can prove them by refering to the module axioms proven in the `module0` context

Vector addition is associative.

```
lemma (in vector_space0) vec_spce_ax1: assumes u∈V v∈V w∈V
  shows u +V (v +V w) = (u +V v) +V w
  using assms vspce_mod.mod_ab_gr.group_oper_assoc by simp
```

Vector addition is commutative.

```
lemma (in vector_space0) vec_spce_ax2: assumes u∈V v∈V
  shows u +V v = v +V u
  using mAbGr assms unfolding IsCommutative_def by simp
```

The zero vector is a vector.

```
lemma (in vector_space0) vec_spce_ax3a: shows Θ ∈ V
  using vspce_mod.zero_in_mod by simp
```

The zero vector is the neutral element of addition of vectors.

```
lemma (in vector_space0) vec_spce_ax3b: assumes v∈V shows v +V Θ =
v
  using assms vspce_mod.zero_neutral by simp
```

The additive inverse of a vector is a vector.

```
lemma (in vector_space0) vec_spce_ax4a: assumes v∈V shows (−v) ∈ V
  using assms vspce_mod.mod_ab_gr.inverse_in_group by simp
```

Sum of of a vector and it's additive inverse is the zero vector.

```
lemma (in vector_space0) vec_spce_ax4b: assumes v∈V
  shows v +V (−v) = Θ
  using assms vspce_mod.mod_ab_gr.group0_2_L6 by simp
```

Scalar multiplication and field multiplication are "compatible" (as Wikipedia calls it).

```
lemma (in vector_space0) vec_spce_ax5: assumes x∈K y∈K v∈V
  shows x·S(y·Sv) = (x·y)·Sv
  using assms vspce_mod.module_ax3 by simp
```

Multiplying the identity element of the field by a vector gives the vector.

```
lemma (in vector_space0) vec_spce_ax6: assumes v∈V shows 1·Sv = v
  using assms vspce_mod.module_ax4 by simp
```

Scalar multiplication is distributive with respect to vector addition.

```
lemma (in vector_space0) vec_spce_ax7: assumes x∈K u∈V v∈V
  shows x·S(u+Vv) = x·Su +V x·Sv
  using assms vspce_mod.module_ax1 by simp
```

Scalar multiplication is distributive with respect to field addition.

lemma (**in** vector_space0) vec_spce_ax8: **assumes** x∈K y∈K v∈V
  shows (x+y)$\cdot_S$v = x$\cdot_S$v $+_V$ y$\cdot_S$v
  using assms vspce_mod.module_ax2 **by** simp

end

54 Ordered fields

theory OrderedField_ZF **imports** OrderedRing_ZF Field_ZF

begin

This theory covers basic facts about ordered fiels.

54.1 Definition and basic properties

Here we define ordered fields and proove their basic properties.

Ordered field is a notrivial ordered ring such that all non-zero elements have an inverse. We define the notion of being a ordered field as a statement about four sets. The first set, denoted K is the carrier of the field. The second set, denoted A represents the additive operation on K (recall that in ZF set theory functions are sets). The third set M represents the multiplicative operation on K. The fourth set r is the order relation on K.

definition
  IsAnOrdField(K,A,M,r) ≡ (IsAnOrdRing(K,A,M,r) ∧
  (M {is commutative on} K) ∧
  TheNeutralElement(K,A) ≠ TheNeutralElement(K,M) ∧
  (∀a∈K. a≠TheNeutralElement(K,A)⟶
  (∃b∈K. M⟨a,b⟩ = TheNeutralElement(K,M))))

The next context (locale) defines notation used for ordered fields. We do that by extending the notation defined in the ring1 context that is used for ordered rings and adding some assumptions to make sure we are talking about ordered fields in this context. We should rename the carrier from R used in the ring1 context to K, more appriopriate for fields. Theoretically the Isar locale facility supports such renaming, but we experienced diffculties using some lemmas from ring1 locale after renaming.

locale field1 = ring1 +

  assumes mult_commute: M {is commutative on} R

  assumes not_triv: **0** ≠ **1**

  assumes inv_exists: ∀a∈R. a≠**0** ⟶ (∃b∈R. a·b = **1**)

```
  fixes non_zero (R₀)
  defines non_zero_def[simp]: R₀ ≡ R-{0}

  fixes inv (_⁻¹ [96] 97)
  defines inv_def[simp]: a⁻¹ ≡ GroupInv(R₀,restrict(M,R₀×R₀))(a)
```

The next lemma assures us that we are talking fields in the `field1` context.

```
lemma (in field1) OrdField_ZF_1_L1: shows IsAnOrdField(R,A,M,r)
  using OrdRing_ZF_1_L1 mult_commute not_triv inv_exists IsAnOrdField_def
  by simp
```

Ordered field is a field, of course.

```
lemma OrdField_ZF_1_L1A: assumes IsAnOrdField(K,A,M,r)
  shows IsAfield(K,A,M)
  using assms IsAnOrdField_def IsAnOrdRing_def IsAfield_def
  by simp
```

Theorems proven in `field0` (about fields) context are valid in the `field1` context (about ordered fields).

```
lemma (in field1) OrdField_ZF_1_L1B: shows field0(R,A,M)
  using OrdField_ZF_1_L1 OrdField_ZF_1_L1A field_field0
  by simp
```

We can use theorems proven in the `field1` context whenever we talk about an ordered field.

```
lemma OrdField_ZF_1_L2: assumes IsAnOrdField(K,A,M,r)
  shows field1(K,A,M,r)
  using assms IsAnOrdField_def OrdRing_ZF_1_L2 ring1_def
    IsAnOrdField_def field1_axioms_def field1_def
  by auto
```

In ordered rings the existence of a right inverse for all positive elements implies the existence of an inverse for all non zero elements.

```
lemma (in ring1) OrdField_ZF_1_L3:
  assumes A1: ∀a∈R₊. ∃b∈R. a·b = 1 and A2: c∈R  c≠0
  shows ∃b∈R. c·b = 1
proof -
  { assume c∈R₊
    with A1 have ∃b∈R. c·b = 1 by simp }
  moreover
  { assume c∉R₊
    with A2 have (-c) ∈ R₊
      using OrdRing_ZF_3_L2A by simp
    with A1 obtain b where b∈R and (-c)·b = 1
      by auto
    with A2 have (-b) ∈ R  c·(-b) = 1
      using Ring_ZF_1_L3 Ring_ZF_1_L7 by auto
```

```
    then have ∃b∈R. c·b = 1 by auto }
  ultimately show thesis by blast
qed
```

Ordered fields are easier to deal with, because it is sufficient to show the existence of an inverse for the set of positive elements.

```
lemma (in ring1) OrdField_ZF_1_L4:
  assumes 0 ≠ 1 and M {is commutative on} R
  and ∀a∈R₊. ∃b∈R. a·b = 1
  shows IsAnOrdField(R,A,M,r)
  using assms OrdRing_ZF_1_L1 OrdField_ZF_1_L3 IsAnOrdField_def
  by simp
```

The set of positive field elements is closed under multiplication.

```
lemma (in field1) OrdField_ZF_1_L5: shows R₊ {is closed under} M
  using OrdField_ZF_1_L1B field0.field_has_no_zero_divs OrdRing_ZF_3_L3
  by simp
```

The set of positive field elements is closed under multiplication: the explicit version.

```
lemma (in field1) pos_mul_closed:
  assumes A1: 0 < a  0 < b
  shows 0 < a·b
proof -
  from A1 have a ∈ R₊ and  b ∈ R₊
    using OrdRing_ZF_3_L14 by auto
  then show 0 < a·b
    using OrdField_ZF_1_L5 IsOpClosed_def PositiveSet_def
    by simp
qed
```

In fields square of a nonzero element is positive.

```
lemma (in field1) OrdField_ZF_1_L6: assumes a∈R  a≠0
  shows a² ∈ R₊
  using assms OrdField_ZF_1_L1B field0.field_has_no_zero_divs
    OrdRing_ZF_3_L15 by simp
```

The next lemma restates the fact from `Field_ZF` that out notation for the field inverse means what it is supposed to mean.

```
lemma (in field1) OrdField_ZF_1_L7: assumes a∈R  a≠0
  shows a·(a⁻¹) = 1  (a⁻¹)·a = 1
  using assms OrdField_ZF_1_L1B field0.Field_ZF_1_L6
  by auto
```

A simple lemma about multiplication and cancelling of a positive field element.

```
lemma (in field1) OrdField_ZF_1_L7A:
```

```
  assumes A1: a∈R  b ∈ R+
  shows
  a·b·b⁻¹ = a
  a·b⁻¹·b = a
proof -
  from A1 have b∈R  b≠0 using PositiveSet_def
    by auto
  with A1 show  a·b·b⁻¹ = a and a·b⁻¹·b = a
    using OrdField_ZF_1_L1B field0.Field_ZF_1_L7
    by auto
qed
```

Some properties of the inverse of a positive element.

```
lemma (in field1) OrdField_ZF_1_L8: assumes A1: a ∈ R+
  shows a⁻¹ ∈ R+  a·(a⁻¹) = 1  (a⁻¹)·a = 1
proof -
  from A1 have I: a∈R  a≠0 using PositiveSet_def
    by auto
  with A1 have a·(a⁻¹)² ∈ R+
    using OrdField_ZF_1_L1B field0.Field_ZF_1_L5 OrdField_ZF_1_L6
      OrdField_ZF_1_L5 IsOpClosed_def by simp
  with I show a⁻¹ ∈ R+
    using OrdField_ZF_1_L1B field0.Field_ZF_2_L1
    by simp
  from I show  a·(a⁻¹) = 1  (a⁻¹)·a = 1
    using OrdField_ZF_1_L7 by auto
qed
```

$\frac{1}{2}$ is a positive member of the field.

```
lemma (in field1) one_half_pos: shows 2⁻¹ ∈ R+ 0<2⁻¹
  using not_triv two_positive OrdField_ZF_1_L8(1) element_pos by simp_all
```

If a is smaller than b, then $(b-a)^{-1}$ is positive.

```
lemma (in field1) OrdField_ZF_1_L9: assumes a<b
  shows  (b-a)⁻¹ ∈ R+
  using assms OrdRing_ZF_1_L14 OrdField_ZF_1_L8
  by simp
```

In ordered fields if at least one of a, b is not zero, then $a^2 + b^2 > 0$, in particular $a^2 + b^2 \neq 0$ and the (multiplicative) inverse of $a^2 + b^2$ exists.

```
lemma (in field1) OrdField_ZF_1_L10:
  assumes A1: a∈R  b∈R and A2: a ≠ 0 ∨ b ≠ 0
  shows 0 < a² + b²  and ∃c∈R. (a² + b²)·c = 1
proof -
  from A1 A2 show 0 < a² + b²
    using OrdField_ZF_1_L1B field0.field_has_no_zero_divs
      OrdRing_ZF_3_L19 by simp
  then have
```

```
    (a² + b²)⁻¹ ∈ R and (a² + b²)·(a² + b²)⁻¹ = 1
    using OrdRing_ZF_1_L3 PositiveSet_def OrdField_ZF_1_L8
    by auto
  then show ∃c∈R. (a² + b²)·c = 1 by auto
qed
```

54.2 Inequalities

In this section we develop tools to deal inequalities in fields.

We can multiply strict inequality by a positive element.

```
lemma (in field1) OrdField_ZF_2_L1:
  assumes a<b and c∈R₊
  shows a·c < b·c
  using assms OrdField_ZF_1_L1B field0.field_has_no_zero_divs
    OrdRing_ZF_3_L13
  by simp
```

A special case of `OrdField_ZF_2_L1` when we multiply an inverse by an element.

```
lemma (in field1) OrdField_ZF_2_L2:
  assumes A1: a∈R₊ and A2: a⁻¹ < b
  shows 1 < b·a
proof -
  from A1 A2 have (a⁻¹)·a < b·a
    using OrdField_ZF_2_L1 by simp
  with A1 show 1 < b·a
    using OrdField_ZF_1_L8 by simp
qed
```

We can multiply an inequality by the inverse of a positive element.

```
lemma (in field1) OrdField_ZF_2_L3:
  assumes a≤b  and c∈R₊ shows a·(c⁻¹) ≤ b·(c⁻¹)
  using assms OrdField_ZF_1_L8 OrdRing_ZF_1_L9A
  by simp
```

We can multiply a strict inequality by a positive element or its inverse.

```
lemma (in field1) OrdField_ZF_2_L4:
  assumes a<b and c∈R₊
  shows
  a·c < b·c
  c·a < c·b
  a·c⁻¹ < b·c⁻¹
   using assms OrdField_ZF_1_L1B field0.field_has_no_zero_divs
    OrdField_ZF_1_L8 OrdRing_ZF_3_L13 by auto
```

We can put a positive factor on the other side of an inequality, changing it to its inverse.

```
lemma (in field1) OrdField_ZF_2_L5:
  assumes A1: a∈R  b∈R₊ and A2: a·b ≤ c
  shows a ≤ c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ ≤ c·b⁻¹
    using OrdField_ZF_2_L3 by simp
  with A1 show a ≤ c·b⁻¹ using OrdField_ZF_1_L7A
    by simp
qed
```

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with a product initially on the right hand side.

```
lemma (in field1) OrdField_ZF_2_L5A:
  assumes A1: b∈R  c∈R₊ and A2: a ≤ b·c
  shows a·c⁻¹ ≤ b
proof -
  from A1 A2 have a·c⁻¹ ≤ b·c·c⁻¹
    using OrdField_ZF_2_L3 by simp
  with A1 show a·c⁻¹ ≤ b using OrdField_ZF_1_L7A
    by simp
qed
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with a product initially on the left hand side.

```
lemma (in field1) OrdField_ZF_2_L6:
  assumes A1: a∈R  b∈R₊ and A2: a·b < c
  shows a < c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ < c·b⁻¹
    using OrdField_ZF_2_L4 by simp
  with A1 show a < c·b⁻¹ using OrdField_ZF_1_L7A
    by simp
qed
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with a product initially on the right hand side.

```
lemma (in field1) OrdField_ZF_2_L6A:
  assumes A1: b∈R  c∈R₊ and A2: a < b·c
  shows a·c⁻¹ < b
proof -
  from A1 A2 have a·c⁻¹ < b·c·c⁻¹
    using OrdField_ZF_2_L4 by simp
  with A1 show a·c⁻¹ < b using OrdField_ZF_1_L7A
    by simp
qed
```

Sometimes we can reverse an inequality by taking inverse on both sides.

```
lemma (in field1) OrdField_ZF_2_L7:
```

```
  assumes A1: a∈R+ and A2: a⁻¹ ≤ b
  shows b⁻¹ ≤ a
proof -
  from A1 have a⁻¹ ∈ R+ using OrdField_ZF_1_L8
    by simp
  with A2 have b ∈ R+ using  OrdRing_ZF_3_L7
    by blast
  then have T: b ∈ R+   b⁻¹ ∈ R+ using OrdField_ZF_1_L8
    by auto
  with A1 A2 have b⁻¹·a⁻¹·a ≤ b⁻¹·b·a
    using OrdRing_ZF_1_L9A by simp
  moreover
  from A1 A2 T have
    b⁻¹ ∈ R  a∈R a≠0  b∈R  b≠0
    using PositiveSet_def OrdRing_ZF_1_L3 by auto
  then have b⁻¹·a⁻¹·a = b⁻¹ and  b⁻¹·b·a = a
    using OrdField_ZF_1_L1B field0.Field_ZF_1_L7
      field0.Field_ZF_1_L6 Ring_ZF_1_L3
    by auto
  ultimately show b⁻¹ ≤ a by simp
qed
```

Sometimes we can reverse a strict inequality by taking inverse on both sides.

```
lemma (in field1) OrdField_ZF_2_L8:
  assumes A1: a∈R+ and A2: a⁻¹ < b
  shows b⁻¹ < a
proof -
  from A1 A2 have a⁻¹ ∈ R+   a⁻¹ ≤b
    using OrdField_ZF_1_L8 by auto
  then have b ∈ R+ using OrdRing_ZF_3_L7
    by blast
  then have b∈R  b≠0 using PositiveSet_def by auto
  with A2 have b⁻¹ ≠ a
    using OrdField_ZF_1_L1B field0.Field_ZF_2_L4
    by simp
  with A1 A2 show b⁻¹ < a
    using OrdField_ZF_2_L7 by simp
qed
```

If the left side of the strict inequality is positive then taking inverses of both sides reverses the inequality.

```
lemma (in field1) poz_elem_inverse_sides: assumes 0<a a<b
  shows b⁻¹ < a⁻¹
proof -
  from assms(1) have a = (a⁻¹)⁻¹
    using OrdRing_ZF_1_L3(2) OrdField_ZF_1_L1B field0.non_zero_inv_inv
by force
  with assms show thesis using element_pos OrdField_ZF_1_L8(1) OrdField_ZF_2_L8
    by simp
```

```
qed
```

A technical lemma about solving a strict inequality with three field elements and inverse of a difference.

```
lemma (in field1) OrdField_ZF_2_L9:
  assumes A1: a<b and A2: (b-a)⁻¹ < c
  shows 1 + a·c < b·c
proof -
  from A1 A2 have (b-a)⁻¹ ∈ R₊  (b-a)⁻¹ ≤ c
    using OrdField_ZF_1_L9 by auto
  then have T1: c ∈ R₊ using OrdRing_ZF_3_L7 by blast
  with A1 A2 have T2:
    a∈R  b∈R  c∈R  c≠0   c⁻¹ ∈ R
    using OrdRing_ZF_1_L3 OrdField_ZF_1_L8 PositiveSet_def
    by auto
  with A1 A2  have c⁻¹ + a < b-a + a
    using OrdRing_ZF_1_L14 OrdField_ZF_2_L8 ring_strict_ord_trans_inv
    by simp
  with T1 T2 have (c⁻¹ + a)·c < b·c
    using Ring_ZF_2_L1A OrdField_ZF_2_L1 by simp
  with T1 T2 show 1 + a·c < b·c
    using ring_oper_distr OrdField_ZF_1_L8
    by simp
qed
```

One half is field member and sum of two halfs is one.

```
lemma (in field1) half_half_one: shows 2⁻¹∈R 2⁻¹+2⁻¹ = 1
proof -
  show 2⁻¹∈R using one_half_pos(2) ord_ring_less_members by simp
  from not_triv have 2∈R and 2≠0 using Ring_ZF_1_L2(4) two_positive(2)
by auto
  then have 2·2⁻¹ = 1 using OrdField_ZF_1_L1B field0.Field_ZF_1_L6
    by simp
  with ‹2⁻¹∈R›  show 2⁻¹+2⁻¹ = 1 using Ring_ZF_1_L3(9) by simp
qed
```

54.3 Definition of real numbers

The only purpose of this section is to define what does it mean to be a model of real numbers.

We define model of real numbers as any quadruple of sets (K, A, M, r) such that (K, A, M, r) is an ordered field and the order relation r is complete, that is every set that is nonempty and bounded above in this relation has a supremum.

```
definition
  IsAmodelOfReals(K,A,M,r) ≡ IsAnOrdField(K,A,M,r) ∧ (r {is complete})
```

end

55 Integers - introduction

theory `Int_ZF_IML` **imports** `OrderedGroup_ZF_1 Finite_ZF_1 ZF.Int Nat_ZF_IML`

begin

This theory file is an interface between the old-style Isabelle (ZF logic) material on integers and the IsarMathLib project. Here we redefine the meta-level operations on integers (addition and multiplication) to convert them to ZF-functions and show that integers form a commutative group with respect to addition and commutative monoid with respect to multiplication. Similarly, we redefine the order on integers as a relation, that is a subset of $Z \times Z$. We show that a subset of intergers is bounded iff it is finite. As we are forced to use standard Isabelle notation with all these dollar signs, sharps etc. to denote "type coercions" (?) the notation is often ugly and difficult to read.

55.1 Addition and multiplication as ZF-functions.

In this section we provide definitions of addition and multiplication as subsets of $(Z \times Z) \times Z$. We use the (higher order) relation defined in the standard `Int` theory to define a subset of $Z \times Z$ that constitutes the ZF order relation corresponding to it. We define the set of positive integers using the notion of positive set from the `OrderedGroup_ZF` theory.

Definition of addition of integers as a binary operation on `int`. Recall that in standard Isabelle/ZF `int` is the set of integers and the sum of integers is denoted by prepending + with a dollar sign.

definition

```
IntegerAddition ≡ { ⟨ x,c⟩ ∈ (int×int)×int. fst(x) $+ snd(x) = c}
```

Definition of multiplication of integers as a binary operation on `int`. In standard Isabelle/ZF product of integers is denoted by prepending the dollar sign to `*`.

definition

```
IntegerMultiplication ≡
  { ⟨ x,c⟩ ∈ (int×int)×int. fst(x) $* snd(x) = c}
```

Definition of natural order on integers as a relation on `int`. In the standard Isabelle/ZF the inequality relation on integers is denoted ≤ prepended with the dollar sign.

definition

```
IntegerOrder ≡ {p ∈ int×int. fst(p) $≤ snd(p)}
```

This defines the set of positive integers.

definition
```
PositiveIntegers ≡ PositiveSet(int,IntegerAddition,IntegerOrder)
```

IntegerAddition and IntegerMultiplication are functions on `int × int`.

lemma `Int_ZF_1_L1:` **shows**
```
  IntegerAddition : int×int → int
  IntegerMultiplication : int×int → int
proof -
  have
    {⟨ x,c⟩ ∈ (int×int)×int. fst(x) $+ snd(x) = c} ∈ int×int→int
    {⟨ x,c⟩ ∈ (int×int)×int. fst(x) $* snd(x) = c} ∈ int×int→int
    using func1_1_L11A by auto
  then show IntegerAddition : int×int → int
    IntegerMultiplication : int×int → int
    using IntegerAddition_def IntegerMultiplication_def by auto
qed
```

The next context (locale) defines notation used for integers. We define **0** to denote the neutral element of addition, **1** as the unit of the multiplicative monoid. We introduce notation `m≤n` for integers and write `m..n` to denote the integer interval with endpoints in m and n. `abs(m)` means the absolute value of m. This is a function defined in `OrderedGroup` that assigns x to itself if x is positive and assigns the opposite of x if $x \leq 0$. Unforunately we cannot use the $|\cdot|$ notation as in the `OrderedGroup` theory as this notation has been hogged by the standard Isabelle's `Int` theory. The notation `-A` where A is a subset of integers means the set $\{-m : m \in A\}$. The symbol `maxf(f,M)` denotes tha maximum of function f over the set A. We also introduce a similar notation for the minimum.

locale `int0 =`
```

  fixes ints (ℤ)
  defines ints_def [simp]: ℤ ≡ int

  fixes ia (infixl + 69)
  defines ia_def [simp]: a+b ≡ IntegerAddition⟨ a,b⟩

  fixes iminus (- _ 72)
  defines rminus_def [simp]: -a ≡ GroupInv(ℤ,IntegerAddition)(a)

  fixes isub (infixl - 69)
  defines isub_def [simp]: a-b ≡ a+ (- b)

  fixes imult (infixl · 70)
  defines imult_def [simp]: a·b ≡ IntegerMultiplication⟨ a,b⟩

  fixes setneg (- _ 72)
```

defines setneg_def [simp]: -A ≡ GroupInv(ℤ,IntegerAddition)(A)

fixes izero (**0**)
defines izero_def [simp]: **0** ≡ TheNeutralElement(ℤ,IntegerAddition)

fixes ione (**1**)
defines ione_def [simp]: **1** ≡ TheNeutralElement(ℤ,IntegerMultiplication)

fixes itwo (**2**)
defines itwo_def [simp]: **2** ≡ **1**+**1**

fixes ithree (**3**)
defines ithree_def [simp]: **3** ≡ **2**+**1**

fixes nonnegative ($\mathbb{Z}^+$)
defines nonnegative_def [simp]:
$\mathbb{Z}^+$ ≡ Nonnegative(ℤ,IntegerAddition,IntegerOrder)

fixes positive ($\mathbb{Z}_+$)
defines positive_def [simp]:
$\mathbb{Z}_+$ ≡ PositiveSet(ℤ,IntegerAddition,IntegerOrder)

fixes abs
defines abs_def [simp]:
abs(m) ≡ AbsoluteValue(ℤ,IntegerAddition,IntegerOrder)(m)

fixes lesseq (**infix** ≤ 60)
defines lesseq_def [simp]: m ≤ n ≡ ⟨m,n⟩ ∈ IntegerOrder

fixes sless (**infix** < 68)
defines sless_def [simp]: a < b ≡ a≤b ∧ a≠b

fixes interval (**infix** .. 70)
defines interval_def [simp]: m..n ≡ Interval(IntegerOrder,m,n)

fixes maxf
defines maxf_def [simp]: maxf(f,A) ≡ Maximum(IntegerOrder,f(A))

fixes minf
defines minf_def [simp]: minf(f,A) ≡ Minimum(IntegerOrder,f(A))

fixes oddext (_ °)
defines oddext_def [simp]: f° ≡ OddExtension(ℤ,IntegerAddition,IntegerOrder,f)

IntegerAddition adds integers and IntegerMultiplication multiplies integers. This states that the ZF functions IntegerAddition and IntegerMultiplication give the same results as the higher-order equivalents defined in the standard Int theory.

lemma (**in** int0) Int_ZF_1_L2: **assumes** A1: a ∈ ℤ b ∈ ℤ

```
  shows
  a+b = a $+ b
  a·b = a $* b
proof -
  let x = ⟨ a,b⟩
  let c = a $+ b
  let d = a $* b
  from A1 have
    ⟨ x,c⟩ ∈ {⟨ x,c⟩ ∈ (ℤ×ℤ)×ℤ. fst(x) $+ snd(x) = c}
    ⟨ x,d⟩ ∈ {⟨ x,d⟩ ∈ (ℤ×ℤ)×ℤ. fst(x) $* snd(x) = d}
    by auto
  then show a+b = a $+ b  a·b = a $* b
    using IntegerAddition_def IntegerMultiplication_def
      Int_ZF_1_L1 apply_iff by auto
qed
```

Integer addition and multiplication are associative.

```
lemma (in int0) Int_ZF_1_L3:
  assumes x∈ℤ  y∈ℤ  z∈ℤ
  shows x+y+z = x+(y+z)  x·y·z = x·(y·z)
  using assms Int_ZF_1_L2 zadd_assoc zmult_assoc by auto
```

Integer addition and multiplication are commutative.

```
lemma (in int0) Int_ZF_1_L4:
  assumes x∈ℤ  y∈ℤ
  shows x+y = y+x  x·y = y·x
  using assms Int_ZF_1_L2 zadd_commute zmult_commute
  by auto
```

Zero is neutral for addition and one for multiplication.

```
lemma (in int0) Int_ZF_1_L5: assumes A1:x∈ℤ
  shows ($# 0) + x = x ∧ x + ($# 0) = x
  ($# 1)·x = x ∧ x·($# 1) = x
proof -
  from A1 show ($# 0) + x = x ∧ x + ($# 0) = x
    using  Int_ZF_1_L2 zadd_int0 Int_ZF_1_L4 by simp
  from A1 have ($# 1)·x = x
    using Int_ZF_1_L2 zmult_int1 by simp
  with A1 show ($# 1)·x = x ∧ x·($# 1) = x
    using Int_ZF_1_L4 by simp
qed
```

Zero is neutral for addition and one for multiplication.

```
lemma (in int0) Int_ZF_1_L6: shows ($# 0)∈ℤ ∧
  (∀x∈ℤ. ($# 0)+x = x ∧ x+($# 0) = x)
  ($# 1)∈ℤ ∧
  (∀x∈ℤ. ($# 1)·x = x ∧ x·($# 1) = x)
  using Int_ZF_1_L5 by auto
```

Integers with addition and integers with multiplication form monoids.

```
theorem (in int0) Int_ZF_1_T1: shows
  IsAmonoid(ℤ,IntegerAddition)
  IsAmonoid(ℤ,IntegerMultiplication)
proof -
   have
    ∃e∈ℤ. ∀x∈ℤ. e+x = x ∧ x+e = x
     ∃e∈ℤ. ∀x∈ℤ. e·x = x ∧ x·e = x
     using int0.Int_ZF_1_L6 by auto
   then show IsAmonoid(ℤ,IntegerAddition)
     IsAmonoid(ℤ,IntegerMultiplication) using
     IsAmonoid_def IsAssociative_def Int_ZF_1_L1 Int_ZF_1_L3
     by auto
qed
```

Zero is the neutral element of the integers with addition and one is the neutral element of the integers with multiplication.

```
lemma (in int0) Int_ZF_1_L8: shows ($# 0) = 0  ($# 1) = 1
proof -
  have monoid0(ℤ,IntegerAddition)
    using Int_ZF_1_T1 monoid0_def by simp
  moreover have
    ($# 0)∈ℤ ∧
    (∀x∈ℤ. IntegerAddition⟨$# 0,x⟩ = x ∧
    IntegerAddition⟨x ,$# 0⟩ = x)
    using Int_ZF_1_L6 by auto
  ultimately have ($# 0) = TheNeutralElement(ℤ,IntegerAddition)
    by (rule monoid0.group0_1_L4)
  then show ($# 0) = 0 by simp
  have monoid0(int,IntegerMultiplication)
    using Int_ZF_1_T1 monoid0_def by simp
  moreover have ($# 1) ∈ int ∧
    (∀x∈int. IntegerMultiplication⟨$# 1, x⟩ = x ∧
    IntegerMultiplication⟨x ,$# 1⟩ = x)
    using Int_ZF_1_L6 by auto
  ultimately have
    ($# 1) = TheNeutralElement(int,IntegerMultiplication)
    by (rule monoid0.group0_1_L4)
  then show  ($# 1) = 1 by simp
qed
```

0 and 1, as defined in `int0` context, are integers.

```
lemma (in int0) Int_ZF_1_L8A: shows 0 ∈ ℤ  1 ∈ ℤ
proof -
  have ($# 0) ∈ ℤ  ($# 1) ∈ ℤ by auto
  then show 0 ∈ ℤ  1 ∈ ℤ using Int_ZF_1_L8 by auto
qed
```

Zero is not one.

```
lemma (in int0) int_zero_not_one: shows 0 ≠ 1
proof -
  have ($# 0) ≠ ($# 1) by simp
  then show 0 ≠ 1 using Int_ZF_1_L8 by simp
qed
```

The set of integers is not empty, of course.

```
lemma (in int0) int_not_empty: shows ℤ ≠ 0
  using Int_ZF_1_L8A by auto
```

The set of integers has more than just zero in it.

```
lemma (in int0) int_not_trivial: shows ℤ ≠ {0}
  using Int_ZF_1_L8A int_zero_not_one by blast
```

Each integer has an inverse (in the addition sense).

```
lemma (in int0) Int_ZF_1_L9: assumes A1: g ∈ ℤ
  shows ∃ b∈ℤ. g+b = 0
proof -
  from A1 have g+ $-g = 0
    using Int_ZF_1_L2 Int_ZF_1_L8 by simp
  thus thesis by auto
qed
```

Integers with addition form an abelian group. This also shows that we can apply all theorems proven in the proof contexts (locales) that require the assumpion that some pair of sets form a group like locale `group0`.

```
theorem Int_ZF_1_T2: shows
  IsAgroup(int,IntegerAddition)
  IntegerAddition {is commutative on} int
  group0(int,IntegerAddition)
  using int0.Int_ZF_1_T1 int0.Int_ZF_1_L9 IsAgroup_def
  group0_def int0.Int_ZF_1_L4 IsCommutative_def by auto
```

Negative of an integer is an integer.

```
lemma (in int0) int_neg_type: assumes m∈ℤ shows (-m) ∈ ℤ
  using assms Int_ZF_1_T2(3) group0.inverse_in_group by simp
```

Taking a negative twice we get back the same integer.

```
lemma (in int0) neg_neg_noop: assumes m∈ℤ shows (-(-m)) = m
  using assms Int_ZF_1_T2(3) group0.group_inv_of_inv by simp
```

What is the additive group inverse in the group of integers?

```
lemma (in int0) Int_ZF_1_L9A: assumes A1: m∈ℤ
  shows $-m = -m
proof -
   from A1 have m∈int $-m ∈ int IntegerAddition⟨ m,$-m⟩ =
     TheNeutralElement(int,IntegerAddition)
```

```
    using zminus_type Int_ZF_1_L2 Int_ZF_1_L8 by auto
  then have $-m = GroupInv(int,IntegerAddition)(m)
    using Int_ZF_1_T2 group0.group0_2_L9 by blast
  then show thesis by simp
qed
```

Subtracting integers corresponds to adding the negative.

```
lemma (in int0) Int_ZF_1_L10: assumes A1: m∈ℤ  n∈ℤ
  shows m-n = m $+ $-n m-n = m $- n
  using assms Int_ZF_1_T2  group0.inverse_in_group Int_ZF_1_L9A Int_ZF_1_L2
zdiff_def
  by simp_all
```

Negative of zero is zero.

```
lemma (in int0) Int_ZF_1_L11: shows (-0) = 0
  using Int_ZF_1_T2  group0.group_inv_of_one by simp
```

A trivial calculation lemma that allows to subtract and add one.

```
lemma Int_ZF_1_L12:
  assumes m∈int shows m $- $#1 $+ $#1 = m
  using assms eq_zdiff_iff by auto
```

A trivial calculation lemma that allows to subtract and add one, version with ZF-operation.

```
lemma (in int0) Int_ZF_1_L13: assumes m∈ℤ
  shows (m $- $#1) + 1 = m
  using assms Int_ZF_1_L8A Int_ZF_1_L2 Int_ZF_1_L8 Int_ZF_1_L12
  by simp
```

Adding or subtracing one changes integers, but subtracting zero does not. .

```
lemma (in int0) Int_ZF_1_L14: assumes A1: m∈ℤ
  shows
  m+1 ≠ m
  m-1 ≠ m
  m-0 = m
proof -
  { assume m+1 = m
    with A1 have
      group0(ℤ,IntegerAddition)
      m∈ℤ  1∈ℤ
      IntegerAddition⟨m,1⟩ = m
      using Int_ZF_1_T2 Int_ZF_1_L8A by auto
    then have 1 = TheNeutralElement(ℤ,IntegerAddition)
      by (rule group0.group0_2_L7)
    then have False using int_zero_not_one by simp
  } then show I: m+1 ≠ m by auto
  { from A1 have m - 1 + 1 = m
      using Int_ZF_1_L8A Int_ZF_1_T2 group0.inv_cancel_two
```

```
    by simp
    moreover assume m-1 = m
    ultimately have m + 1 = m by simp
    with I have False by simp
  } then show m-1 ≠ m by auto
  from assms show  m-0 = m using Int_ZF_1_T2(3) group0.div_by_neutral
    by simp
qed
```

If the difference is zero, the integers are equal.

```
lemma (in int0) Int_ZF_1_L15:
  assumes A1: m∈ℤ  n∈ℤ and A2: m-n = 0
  shows m=n
proof -
  let G = ℤ
  let f = IntegerAddition
  from A1 A2 have
    group0(G, f)
    m ∈ G  n ∈ G
    f⟨m, GroupInv(G, f)(n)⟩ = TheNeutralElement(G, f)
    using Int_ZF_1_T2 by auto
  then show m=n by (rule group0.group0_2_L11A)
qed
```

55.2 Integers as an ordered group

In this section we define order on integers as a relation, that is a subset of $Z \times Z$ and show that integers form an ordered group.

The next lemma interprets the order definition one way.

```
lemma (in int0) Int_ZF_2_L1:
  assumes A1: m∈ℤ n∈ℤ and A2: m $≤ n
  shows m ≤ n
proof -
  from A1 A2 have ⟨ m,n⟩ ∈ {x∈ℤ×ℤ. fst(x) $≤ snd(x)}
    by simp
  then show thesis using IntegerOrder_def by simp
qed
```

The next lemma interprets the definition the other way.

```
lemma (in int0) Int_ZF_2_L1A: assumes A1: m ≤ n
  shows m $≤ n m∈ℤ n∈ℤ
proof -
  from A1 have ⟨ m,n⟩ ∈ {p∈ℤ×ℤ. fst(p) $≤ snd(p)}
    using IntegerOrder_def by simp
  thus m $≤ n  m∈ℤ  n∈ℤ by auto
qed
```

Integer order is a relation on integers.

```
lemma Int_ZF_2_L1B: shows IntegerOrder ⊆ int×int
proof
  fix x assume x∈IntegerOrder
  then have x ∈ {p∈int×int. fst(p) $≤ snd(p)}
    using IntegerOrder_def by simp
  then show x∈int×int by simp
qed
```

The way we define the notion of being bounded below, its sufficient for the relation to be on integers for all bounded below sets to be subsets of integers.

```
lemma (in int0) Int_ZF_2_L1C:
  assumes A1: IsBoundedBelow(A,IntegerOrder)
  shows A⊆ℤ
proof -
  from A1 have
    IntegerOrder ⊆ ℤ×ℤ
    IsBoundedBelow(A,IntegerOrder)
    using Int_ZF_2_L1B by auto
  then show A⊆ℤ by (rule Order_ZF_3_L1B)
qed
```

The order on integers is reflexive.

```
lemma (in int0) int_ord_is_refl: shows refl(ℤ,IntegerOrder)
  using Int_ZF_2_L1 zle_refl refl_def by auto
```

A more explicit version of "integer order is reflexive" claim

```
lemma (in int0) int_ord_is_refl1: assumes z∈ℤ
  shows z≤z
  using assms int_ord_is_refl unfolding refl_def by simp
```

The essential condition to show antisymmetry of the order on integers.

```
lemma (in int0) Int_ZF_2_L3:
  assumes A1: m ≤ n  n ≤ m
  shows m=n
proof -
  from A1 have m $≤ n  n $≤ m  m∈ℤ  n∈ℤ
    using Int_ZF_2_L1A by auto
  then show m=n using zle_anti_sym by auto
qed
```

The order on integers is antisymmetric.

```
lemma (in int0) Int_ZF_2_L4: shows antisym(IntegerOrder)
proof -
  have ∀m n. m ≤ n  ∧ n ≤ m ⟶ m=n
    using Int_ZF_2_L3 by auto
  then show thesis using imp_conj antisym_def by simp
qed
```

The essential condition to show that the order on integers is transitive.

```
lemma Int_ZF_2_L5:
  assumes A1: ⟨m,n⟩ ∈ IntegerOrder  ⟨n,k⟩ ∈ IntegerOrder
  shows ⟨m,k⟩ ∈ IntegerOrder
proof -
  from A1 have T1: m $≤ n n $≤ k and T2: m∈int k∈int
    using int0.Int_ZF_2_L1A by auto
  from T1 have m $≤ k by (rule zle_trans)
  with T2 show thesis using int0.Int_ZF_2_L1 by simp
qed
```

The order on integers is transitive. This version is stated in the `int0` context using notation for integers.

```
lemma (in int0) Int_order_transitive:
  assumes A1: m≤n  n≤k
  shows m≤k
proof -
  from A1 have ⟨ m,n⟩ ∈ IntegerOrder  ⟨ n,k⟩ ∈ IntegerOrder
    by auto
  then have ⟨ m,k⟩ ∈ IntegerOrder by (rule Int_ZF_2_L5)
  then show m≤k by simp
qed
```

The order on integers is transitive.

```
lemma Int_ZF_2_L6: shows trans(IntegerOrder)
proof -
  have ∀ m n k.
    ⟨m, n⟩ ∈ IntegerOrder ∧ ⟨n, k⟩ ∈ IntegerOrder ⟶
    ⟨m, k⟩ ∈ IntegerOrder
    using Int_ZF_2_L5 by blast
  then show thesis by (rule Fol1_L2)
qed
```

The order on integers is a partial order.

```
lemma Int_ZF_2_L7: shows IsPartOrder(int,IntegerOrder)
  using int0.int_ord_is_refl int0.Int_ZF_2_L4
    Int_ZF_2_L6 IsPartOrder_def by simp
```

The essential condition to show that the order on integers is preserved by translations.

```
lemma (in int0) int_ord_transl_inv:
  assumes A1: k ∈ ℤ and A2: m ≤ n
  shows m+k ≤ n+k   k+m≤ k+n
proof -
  from A2 have m $≤ n and m∈ℤ n∈ℤ
    using Int_ZF_2_L1A by auto
  with A1 show m+k ≤ n+k   k+m≤ k+n
    using zadd_right_cancel_zle zadd_left_cancel_zle
    Int_ZF_1_L2 Int_ZF_1_L1 apply_funtype
```

```
    Int_ZF_1_L2 Int_ZF_2_L1 Int_ZF_1_L2 by auto
qed
```

Integers form a linearly ordered group. We can apply all theorems proven in group3 context to integers.

```
theorem (in int0) Int_ZF_2_T1: shows
  IsAnOrdGroup(ℤ,IntegerAddition,IntegerOrder)
  IntegerOrder {is total on} ℤ
  group3(ℤ,IntegerAddition,IntegerOrder)
  IsLinOrder(ℤ,IntegerOrder)
proof -
  have ∀k∈ℤ. ∀m n. m ≤ n  ⟶
    m+k ≤ n+k ∧ k+m≤ k+n
    using int_ord_transl_inv by simp
  then show T1: IsAnOrdGroup(ℤ,IntegerAddition,IntegerOrder) using
    Int_ZF_1_T2 Int_ZF_2_L1B Int_ZF_2_L7 IsAnOrdGroup_def
    by simp
  then show group3(ℤ,IntegerAddition,IntegerOrder)
    using group3_def by simp
  have ∀n∈ℤ. ∀m∈ℤ. n≤m ∨ m≤n
    using zle_linear Int_ZF_2_L1 by auto
  then show IntegerOrder {is total on} ℤ
    using IsTotal_def by simp
  with T1 show IsLinOrder(ℤ,IntegerOrder)
    using IsAnOrdGroup_def IsPartOrder_def IsLinOrder_def by simp
qed
```

Another way of stating that we can apply theorems proven in the `group3` context (defined `OrderedGroup_ZF` theory) to the ordered group of integers.

```
sublocale int0 < group3 int IntegerAddition IntegerOrder
  0 ia iminus lesseq sless nonnegative positive setneg abs oddext
  using Int_ZF_2_T1(3) by auto
```

Negative numbers are not nonnegative. This is a special case of `ls_not_leq` from `OrderedGroup_ZF` theory.

```
corollary (in int0) neg_not_nonneg: assumes m<0 shows ¬(0≤m)
  using assms ls_not_leq by simp
```

Negative of a positive integer is negative.

```
lemma (in int0) neg_pos_int_neg: assumes 0<z shows (-z)<0
  using assms inv_both_strict_ineq Int_ZF_1_L11 by force
```

Negative of a negative integer is positive.

```
lemma (in int0) neg_neg_int_pos: assumes z<0 shows 0<(-z)
  using assms inv_both_strict_ineq Int_ZF_1_L11 by force
```

An integer is nonnegative or negative. This is a special case of `OrdGroup_2cases` from `OrderedGroup_ZF` theory and useful for splitting proofs into cases.

```
lemma (in int0) int_nonneg_or_neg: assumes z∈ℤ shows 0≤z ∨ z<0
  using assms OrdGroup_2cases Int_ZF_1_L8A(1) Int_ZF_2_T1(2) by simp
```

Slightly weaker assertion than `int_nonneg_or_neg` with overlap at zero: an integer is nonnegative or nonpositive.

```
corollary (in int0) int_nonneg_or_nonpos: assumes z∈ℤ shows 0≤z ∨ z≤0
  using assms int_nonneg_or_neg by auto
```

Another variant of splitting into cases: an integer is positive, zero or negative.

```
lemma (in int0) int_neg_zero_pos: assumes z∈ℤ shows 0<z ∨ z=0 ∨ z<0
  using assms OrdGroup_3cases Int_ZF_1_L8A(1) Int_ZF_2_T1(2) by auto
```

If a pair (i, m) belongs to the order relation on integers and $i \neq m$, then i is smaller that m in the sense of defined in the standard Isabelle's `Int.thy`.

```
lemma (in int0) Int_ZF_2_L9: assumes A1: i ≤ m and A2: i≠m
  shows i $< m
proof -
  from A1 have i $≤ m  i∈ℤ  m∈ℤ
    using Int_ZF_2_L1A by auto
  with A2 show i $< m using zle_def by simp
qed
```

This shows how Isabelle's `$<` operator translates to IsarMathLib notation.

```
lemma (in int0) Int_ZF_2_L9AA: assumes A1: m∈ℤ  n∈ℤ
  and A2: m $< n
  shows m≤n  m ≠ n m<n
  using assms zle_def Int_ZF_2_L1 by auto
```

A small technical lemma about putting one on the other side of an inequality.

```
lemma (in int0) Int_ZF_2_L9A:
  assumes A1: k∈ℤ and A2: m ≤ k $- ($# 1)
  shows m+1 ≤ k
proof -
  from A2 have m+1 ≤ (k $- ($# 1)) + 1
    using Int_ZF_1_L8A int_ord_transl_inv by simp
  with A1 show m+1 ≤ k
    using Int_ZF_1_L13 by simp
qed
```

We can put any integer on the other side of an inequality reversing its sign.

```
lemma (in int0) Int_ZF_2_L9B: assumes i∈ℤ  m∈ℤ  k∈ℤ
  shows i+m ≤ k  ⟷ i ≤ k-m
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L9A
  by simp
```

A special case of `Int_ZF_2_L9B` with weaker assumptions.

```
lemma (in int0) Int_ZF_2_L9C:
```

```
  assumes i∈ℤ  m∈ℤ and i-m ≤ k
  shows i ≤ k+m
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L9B
  by simp
```

Taking (higher order) minus on both sides of inequality reverses it.

```
lemma (in int0) Int_ZF_2_L10: assumes k ≤ i
  shows
  (-i) ≤ (-k)
  $-i ≤ $-k
  using assms Int_ZF_2_L1A Int_ZF_1_L9A Int_ZF_2_T1
    group3.OrderedGroup_ZF_1_L5 by auto
```

Taking minus on both sides of inequality reverses it, version with a negative on one side.

```
lemma (in int0) Int_ZF_2_L10AA: assumes n∈ℤ  m≤(-n)
  shows n≤(-m)
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5AD
  by simp
```

We can cancel the same element on on both sides of an inequality, a version with minus on both sides.

```
lemma (in int0) Int_ZF_2_L10AB:
  assumes m∈ℤ  n∈ℤ  k∈ℤ and m-n ≤ m-k
  shows k≤n
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5AF
  by simp
```

If an integer is nonpositive, then its opposite is nonnegative.

```
lemma (in int0) Int_ZF_2_L10A: assumes k ≤ 0
  shows 0≤(-k)
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5A by simp
```

If the opposite of an integers is nonnegative, then the integer is nonpositive.

```
lemma (in int0) Int_ZF_2_L10B:
  assumes k∈ℤ and 0≤(-k)
  shows k≤0
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5AA by simp
```

Adding one to an integer corresponds to taking a successor for a natural number.

```
lemma (in int0) Int_ZF_2_L11:
  shows i $+ $# n $+ ($# 1)  =  i $+ $# succ(n)
proof -
  have $# succ(n) = $#1 $+ $# n using int_succ_int_1 by blast
  then have i $+ $# succ(n) = i $+ ($# n  $+ $#1)
    using zadd_commute by simp
```

```
  then show thesis using zadd_assoc by simp
qed
```

Adding a natural number increases integers.

```
lemma (in int0) Int_ZF_2_L12: assumes A1: i∈ℤ and A2: n∈nat
  shows i ≤ i $+ $#n
proof -
  { assume n = 0
    with A1 have i ≤ i $+ $#n using zadd_int0 int_ord_is_refl refl_def
      by simp }
  moreover
  { assume n≠0
    with A2 obtain k where k∈nat n = succ(k)
      using Nat_ZF_1_L3 by auto
    with A1 have i ≤ i $+ $#n
      using zless_succ_zadd zless_imp_zle Int_ZF_2_L1 by simp }
  ultimately show thesis by blast
qed
```

Adding one increases integers.

```
lemma (in int0) Int_ZF_2_L12A: assumes A1: j≤k
  shows j ≤ k $+ $#1  j ≤ k+1
proof -
  from A1 have T1:j∈ℤ k∈ℤ j $≤ k
    using Int_ZF_2_L1A by auto
  moreover from T1 have k $≤ k $+ $#1 using Int_ZF_2_L12 Int_ZF_2_L1A
    by simp
  ultimately have j $≤ k $+ $#1 using zle_trans by fast
  with T1 show j ≤ k $+ $#1 using Int_ZF_2_L1 by simp
  with T1 have j≤ k+$#1
    using Int_ZF_1_L2 by simp
  then show j ≤ k+1 using Int_ZF_1_L8 by simp
qed
```

Adding one increases integers, yet one more version.

```
lemma (in int0) Int_ZF_2_L12B: assumes A1: m∈ℤ shows m ≤ m+1
  using assms int_ord_is_refl refl_def Int_ZF_2_L12A by simp
```

If $k + 1 = m + n$, where n is a non-zero natural number, then $m \leq k$.

```
lemma (in int0) Int_ZF_2_L13:
  assumes A1: k∈ℤ m∈ℤ and A2: n∈nat
  and A3: k $+ ($# 1) = m $+ $# succ(n)
  shows m ≤ k
proof -
  from A1 have k∈ℤ m $+ $# n ∈ ℤ by auto
  moreover from assms have k $+ $# 1 = m $+ $# n $+ $#1
    using Int_ZF_2_L11 by simp
  ultimately have k = m $+ $# n using zadd_right_cancel by simp
```

```
  with A1 A2 show thesis using Int_ZF_2_L12 by simp
qed
```

The absolute value of an integer is an integer.

```
lemma (in int0) Int_ZF_2_L14: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ
proof -
  have AbsoluteValue(ℤ,IntegerAddition,IntegerOrder) : ℤ→ℤ
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L1 by simp
  with A1 show thesis using apply_funtype by simp
qed
```

If two integers are nonnegative, then the opposite of one is less or equal than the other and the sum is also nonnegative.

```
lemma (in int0) Int_ZF_2_L14A:
  assumes 0≤m  0≤n
  shows
  (-m) ≤ n
  0 ≤ m + n
  using assms Int_ZF_2_T1
    group3.OrderedGroup_ZF_1_L5AC group3.OrderedGroup_ZF_1_L12
      by auto
```

We can increase components in an estimate.

```
lemma (in int0) Int_ZF_2_L15:
  assumes b≤b₁ c≤c₁ and a≤b+c
  shows a≤b₁+c₁
proof -
  from assms have group3(ℤ,IntegerAddition,IntegerOrder)
    ⟨a,IntegerAddition⟨ b,c⟩⟩ ∈ IntegerOrder
    ⟨b,b₁⟩ ∈ IntegerOrder ⟨c,c₁⟩ ∈ IntegerOrder
    using Int_ZF_2_T1 by auto
  then have ⟨a,IntegerAddition⟨ b₁,c₁⟩⟩ ∈ IntegerOrder
    by (rule group3.OrderedGroup_ZF_1_L5E)
  thus thesis by simp
qed
```

We can add or subtract the sides of two inequalities.

```
lemma (in int0) int_ineq_add_sides:
  assumes a≤b and c≤d
  shows
  a+c ≤ b+d
  a-d ≤ b-c
  using assms Int_ZF_2_T1
    group3.OrderedGroup_ZF_1_L5B group3.OrderedGroup_ZF_1_L5I
  by auto
```

We can increase the second component in an estimate.

```
lemma (in int0) Int_ZF_2_L15A:
  assumes b∈ℤ and a≤b+c and A3: c≤c₁
  shows a≤b+c₁
proof -
  from assms have
    group3(ℤ,IntegerAddition,IntegerOrder)
    b ∈ ℤ
    ⟨a,IntegerAddition⟨ b,c⟩⟩ ∈ IntegerOrder
    ⟨c,c₁⟩ ∈ IntegerOrder
    using Int_ZF_2_T1 by auto
  then have ⟨a,IntegerAddition⟨ b,c₁⟩⟩ ∈ IntegerOrder
     by (rule group3.OrderedGroup_ZF_1_L5D)
   thus thesis by simp
qed
```

If we increase the second component in a sum of three integers, the whole sum inceases.

```
lemma (in int0) Int_ZF_2_L15C:
  assumes A1: m∈ℤ  n∈ℤ and A2: k ≤ L
  shows m+k+n ≤ m+L+n
proof -
  let P = IntegerAddition
  from assms have
    group3(int,P,IntegerOrder)
    m ∈ int  n ∈ int
    ⟨k,L⟩ ∈ IntegerOrder
    using Int_ZF_2_T1 by auto
  then have ⟨P⟨P⟨ m,k⟩,n⟩, P⟨P⟨ m,L⟩,n⟩ ⟩ ∈ IntegerOrder
    by (rule group3.OrderedGroup_ZF_1_L10)
  then show m+k+n ≤ m+L+n by simp
qed
```

We don't decrease an integer by adding a nonnegative one.

```
lemma (in int0) Int_ZF_2_L15D:
  assumes 0≤n  m∈ℤ
  shows m ≤ n+m
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5F
  by simp
```

Some inequalities about the sum of two integers and its absolute value.

```
lemma (in int0) Int_ZF_2_L15E:
  assumes m∈ℤ  n∈ℤ
  shows
  m+n ≤ abs(m)+abs(n)
  m-n ≤ abs(m)+abs(n)
  (-m)+n ≤ abs(m)+abs(n)
  (-m)-n ≤ abs(m)+abs(n)
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L6A
  by auto
```

We can add a nonnegative integer to the right hand side of an inequality.

```
lemma (in int0) Int_ZF_2_L15F:  assumes m≤k  and 0≤n
  shows m ≤ k+n  m ≤ n+k
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5G
  by auto
```

Triangle inequality for integers.

```
lemma (in int0) Int_triangle_ineq:
  assumes m∈ℤ  n∈ℤ
  shows abs(m+n)≤abs(m)+abs(n)
  using assms Int_ZF_1_T2 Int_ZF_2_T1 group3.OrdGroup_triangle_ineq
  by simp
```

Taking absolute value does not change nonnegative integers.

```
lemma (in int0) Int_ZF_2_L16:
  assumes 0≤m shows  m∈ℤ⁺ and abs(m) = m
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L2
    group3.OrderedGroup_ZF_3_L2 by auto
```

$0 \leq 1$, so $|1| = 1$.

```
lemma (in int0) Int_ZF_2_L16A:
  shows 0≤1 0<1 abs(1) = 1
proof -
  have ($# 0) ∈ ℤ ($# 1)∈ ℤ by auto
  then have 0≤0 and T1: 1∈ℤ
    using Int_ZF_1_L8 int_ord_is_refl refl_def by auto
  then have 0≤0+1 using Int_ZF_2_L12A by simp
  with T1 show 0≤1 using Int_ZF_1_T2 group0.group0_2_L2
    by simp
  then show abs(1) = 1 and 0<1
    using Int_ZF_2_L16 int_zero_not_one by simp_all
qed
```

Negative one is smaller than zero.

```
lemma (in int0) neg_one_less_zero: shows (-1)<0
  using Int_ZF_2_L16A(2) pos_inv_neg by simp
```

$1 \leq 2$.

```
lemma (in int0) Int_ZF_2_L16B: shows 1≤2
proof -
  have ($# 1)∈ ℤ by simp
  then show 1≤2
    using Int_ZF_1_L8 int_ord_is_refl refl_def Int_ZF_2_L12A
    by simp
qed
```

Assume an integer is greater or equal one. Then it is greater or equal than zero, is not equal zero, if we add one to it then it is greater or equal one, two and zero. Two is .

```
lemma (in int0) Int_ZF_2_L16C:
  assumes A1: 1≤a shows
  0≤a  a≠0
  2 ≤ a+1
  1 ≤ a+1
  0 ≤ a+1
proof -
  from A1 have 0≤1 and 1≤a
    using Int_ZF_2_L16A by auto
  then show 0≤a by (rule Int_order_transitive)
  have I: 0≤1 using Int_ZF_2_L16A by simp
  have 1≤2 using Int_ZF_2_L16B by simp
  moreover from A1 show 2 ≤ a+1
    using Int_ZF_1_L8A int_ord_transl_inv by simp
  ultimately show 1 ≤ a+1 by (rule Int_order_transitive)
  with I show 0 ≤ a+1 by (rule Int_order_transitive)
  from A1 show a≠0 using
    Int_ZF_2_L16A Int_ZF_2_L3 int_zero_not_one by auto
qed
```

If we add one to a nonnegative integer, the result is greater than zero.

```
lemma (in int0) nneg_add_one: assumes 0≤a
  shows 0<a+1
proof -
  from assms have 0+0 < a+1
    using Int_ZF_2_L16A(2) OrderedGroup_ZF_1_L12D by simp
  then show 0<a+1 using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

Absolute value is the same for an integer and its opposite.

```
lemma (in int0) Int_ZF_2_L17:
  assumes m∈ℤ shows abs(-m) = abs(m)
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L7A by simp
```

The absolute value of zero is zero.

```
lemma (in int0) Int_ZF_2_L18: shows abs(0) = 0
  using Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L2A by simp
```

A different version of the triangle inequality.

```
lemma (in int0) Int_triangle_ineq1:
  assumes A1: m∈ℤ  n∈ℤ
  shows
  abs(m-n) ≤ abs(n)+abs(m)
  abs(m-n) ≤ abs(m)+abs(n)
proof -
  have $-n ∈ ℤ by simp
  with A1 have abs(m-n) ≤ abs(m)+abs(-n)
```

```
    using Int_ZF_1_L9A Int_triangle_ineq by simp
  with A1 show
    abs(m-n) ≤ abs(n)+abs(m)
    abs(m-n) ≤ abs(m)+abs(n)
    using Int_ZF_2_L17 Int_ZF_2_L14 Int_ZF_1_T2 IsCommutative_def
    by auto
qed
```

Another version of the triangle inequality.

```
lemma (in int0) Int_triangle_ineq2:
  assumes m∈ℤ  n∈ℤ
  and abs(m-n) ≤ k
  shows
  abs(m) ≤ abs(n)+k
  m-k ≤ n
  m ≤ n+k
  n-k ≤ m
  using assms Int_ZF_1_T2 Int_ZF_2_T1
    group3.OrderedGroup_ZF_3_L7D group3.OrderedGroup_ZF_3_L7E
  by auto
```

Triangle inequality with three integers. We could use `OrdGroup_triangle_ineq3`, but since simp cannot translate the notation directly, it is simpler to reprove it for integers.

```
lemma (in int0) Int_triangle_ineq3:
  assumes A1: m∈ℤ  n∈ℤ  k∈ℤ
  shows abs(m+n+k) ≤ abs(m)+abs(n)+abs(k)
proof -
  from A1 have T: m+n ∈ ℤ  abs(k) ∈ ℤ
    using Int_ZF_1_T2 group0.group_op_closed  Int_ZF_2_L14
    by auto
  with A1 have abs(m+n+k) ≤ abs(m+n) + abs(k)
    using Int_triangle_ineq by simp
  moreover from A1 T have
    abs(m+n) + abs(k) ≤ abs(m) + abs(n) + abs(k)
    using Int_triangle_ineq int_ord_transl_inv by simp
  ultimately show thesis by (rule Int_order_transitive)
qed
```

The next lemma shows what happens when one integers is not greater or equal than another.

```
lemma (in int0) Int_ZF_2_L19:
  assumes A1: m∈ℤ  n∈ℤ and A2: ¬(n≤m)
  shows m≤n  (-n) ≤ (-m)  m≠n m<n
  using OrderedGroup_ZF_1_L8 Int_ZF_2_T1 assms by auto
```

If one integer is greater or equal and not equal to another, then it is not smaller or equal.

```
lemma (in int0) Int_ZF_2_L19AA: assumes A1: m≤n and A2: m≠n
  shows ¬(n≤m)
proof -
  from A1 A2 have
    group3(ℤ, IntegerAddition, IntegerOrder)
    ⟨m,n⟩ ∈ IntegerOrder
    m≠n
    using Int_ZF_2_T1 by auto
  then have ⟨n,m⟩ ∉ IntegerOrder
    by (rule group3.OrderedGroup_ZF_1_L8AA)
  thus ¬(n≤m) by simp
qed
```

The next lemma allows to prove theorems for the case of positive and negative integers separately.

```
lemma (in int0) Int_ZF_2_L19A: assumes A1: m∈ℤ and A2: ¬(0≤m)
  shows m≤0  0 ≤ (-m)  m≠0
proof -
  from A1 have T: 0 ∈ ℤ
    using Int_ZF_1_T2 group0.group0_2_L2 by auto
  with A1 A2 show m≤0 using Int_ZF_2_L19 by blast
  from A1 T A2 show m≠0  by (rule Int_ZF_2_L19)
  from A1 T A2 have (-0)≤(-m) by (rule Int_ZF_2_L19)
  then show 0 ≤ (-m)
    using Int_ZF_1_T2 group0.group_inv_of_one by simp
qed
```

We can prove a theorem about integers by proving that it holds for $m = 0$, $m \in \mathbb{Z}_+$ and $-m \in \mathbb{Z}_+$.

```
lemma (in int0) Int_ZF_2_L19B:
  assumes m∈ℤ and Q(0) and ∀n∈ℤ+. Q(n) and ∀n∈ℤ+. Q(-n)
  shows Q(m)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  let b = m
  from assms have
    group3(G, P, r)
    r {is total on} G
    b ∈ G
    Q(TheNeutralElement(G, P))
    ∀a∈PositiveSet(G, P, r). Q(a)
    ∀a∈PositiveSet(G, P, r). Q(GroupInv(G, P)(a))
    using Int_ZF_2_T1 by auto
  then show Q(b) by (rule group3.OrderedGroup_ZF_1_L18)
qed
```

An integer is not greater than its absolute value.

```
lemma (in int0) Int_ZF_2_L19C: assumes A1: m∈ℤ
  shows
  m ≤ abs(m)
  (-m) ≤ abs(m)
  using assms Int_ZF_2_T1
    group3.OrderedGroup_ZF_3_L5 group3.OrderedGroup_ZF_3_L6
  by auto
```

$|m - n| = |n - m|$.

```
lemma (in int0) Int_ZF_2_L20: assumes m∈ℤ  n∈ℤ
  shows abs(m-n) = abs(n-m)
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L7B by simp
```

We can add the sides of inequalities with absolute values.

```
lemma (in int0) Int_ZF_2_L21:
  assumes A1: m∈ℤ n∈ℤ
  and A2: abs(m) ≤ k  abs(n) ≤ l
  shows
  abs(m+n) ≤ k + l
  abs(m-n) ≤ k + l
  using assms Int_ZF_1_T2 Int_ZF_2_T1
    group3.OrderedGroup_ZF_3_L7C group3.OrderedGroup_ZF_3_L7CA
  by auto
```

Absolute value is nonnegative.

```
lemma (in int0) int_abs_nonneg: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ⁺  0 ≤ abs(m)
proof -
  have AbsoluteValue(ℤ,IntegerAddition,IntegerOrder) : ℤ→ℤ⁺
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L3C by simp
  with A1 show abs(m) ∈ ℤ⁺ using apply_funtype
    by simp
  then show 0 ≤ abs(m)
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L2 by simp
qed
```

If an nonnegative integer is less or equal than another, then so is its absolute value.

```
lemma (in int0) Int_ZF_2_L23:
  assumes 0≤m   m≤k
  shows abs(m) ≤ k
  using assms Int_ZF_2_L16 by simp
```

The standard Isabelle/ZF defined `znegative` predicate on integers. The next lemma expresses that in terms of the order relation on integers.

```
lemma (in int0) znegative_as_ls_zero: assumes z∈ℤ
  shows znegative(z) ⟷ z<0
proof
```

```
  assume znegative(z)
  with assms show z<0
    using Int_ZF_1_L14 Int_ZF_1_L8A(1) Int_ZF_1_L10(2) Int_ZF_2_L9AA(3)
    unfolding zless_def by simp
next
  assume z<0
  with assms have znegative(z-0)
    using Int_ZF_1_L8A(1) Int_ZF_2_L9 Int_ZF_1_L10(2)
    unfolding zless_def by simp
  with assms show znegative(z) using Int_ZF_1_L14 by simp
qed
```

A nonnegative integer (i.e. $0 \leq z$) is not negative in the sense of `znegative` predicate. We also use the opportunity to mention that such a nonnegative integer is an integer.

```
lemma (in int0) nonnegative_not_znegative: assumes 0≤z
  shows ¬znegative(z) and z∈ℤ
proof -
  from assms show z∈ℤ using OrderedGroup_ZF_1_L4(2) by simp
  { assume znegative(z)
    with ‹z∈ℤ› have z<0 using znegative_as_ls_zero by simp
    with assms have 0<0 using group_strict_ord_transit by blast
    hence False by simp
  } thus ¬znegative(z) by auto
qed
```

A nonnegative integer (i.e. one that belongs to $\mathbb{Z}^+$) is not negative in the sense of `znegative` predicate. We also use the opportunity to mention that a nonnegative integer is an integer.

```
lemma (in int0) nonnegative_not_znegative1: assumes z∈ℤ⁺
  shows ¬znegative(z) and z∈ℤ
  using assms OrderedGroup_ZF_1_L2 nonnegative_not_znegative by simp_all
```

A negative integer is `znegative`.

```
lemma (in int0) negative_is_znegative: assumes z∈ℤ\ℤ⁺ shows znegative(z)
  using assms OrderedGroup_ZF_1_L2 Int_ZF_2_T1(2) Int_ZF_1_L8A(1)
    OrderedGroup_ZF_1_L8(4) znegative_as_ls_zero by simp
```

An integer that is not `znegative` is nonegative.

```
corollary (in int0) notzneg_is_nonneg: assumes z∈ℤ and ¬znegative(z)
  shows z∈ℤ⁺ using assms negative_is_znegative by auto
```

An integers that is not `znegative` is greater or equal than zero..

```
lemma (in int0) notzneg_is_geq_zero: assumes z∈ℤ and ¬znegative(z)
  shows 0≤z
  using assms notzneg_is_nonneg Nonnegative_def by simp
```

55.3 Induction on integers.

In this section we show some induction lemmas for integers. The basic tools are the induction on natural numbers and the fact that integers can be written as a sum of a smaller integer and a natural number.

An integer can be written a a sum of a smaller integer and a natural number.

```
lemma (in int0) Int_ZF_3_L2: assumes A1: i ≤ m
  shows ∃n∈nat. m = i $+ $# n
proof -
  let n = 0
  { assume A2: i=m
    from A1 A2 have n ∈ nat m = i $+ $# n
      using Int_ZF_2_L1A zadd_int0_right by auto
    hence ∃n∈nat. m = i $+ $# n by blast }
  moreover
  { assume A3: i≠m
    with A1 have i $< m i∈ℤ m∈ℤ
      using Int_ZF_2_L9 Int_ZF_2_L1A by auto
    then obtain k where D1: k∈nat m = i $+ $# succ(k)
      using zless_imp_succ_zadd_lemma by auto
    let n = succ(k)
    from D1 have n∈nat m = i $+ $# n by auto
    hence ∃n∈nat. m = i $+ $# n by simp }
  ultimately show thesis by blast
qed
```

Induction for integers, the induction step.

```
lemma (in int0) Int_ZF_3_L6: assumes A1: i∈ℤ
  and A2: ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
  shows ∀k∈nat. Q(i $+ ($# k)) ⟶ Q(i $+ ($# succ(k)))
proof
  fix k assume A3: k∈nat show Q(i $+ $# k) ⟶ Q(i $+ $# succ(k))
  proof
    assume A4: Q(i $+ $# k)
    from A1 A3 have i≤ i $+ ($# k) using Int_ZF_2_L12
      by simp
    with A4 A2 have Q(i $+ ($# k) $+ ($# 1)) by simp
    then show Q(i $+ ($# succ(k))) using Int_ZF_2_L11 by simp
  qed
qed
```

Induction on integers, version with higher-order increment function.

```
lemma (in int0) Int_ZF_3_L7:
  assumes A1: i≤k and A2: Q(i)
  and A3: ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
  shows Q(k)
proof -
  from A1 obtain n where D1: n∈nat and D2: k = i $+ $# n
```

```
    using Int_ZF_3_L2 by auto
  from A1 have T1: i∈ℤ using Int_ZF_2_L1A by simp
  note ‹n∈nat›
  moreover from A1 A2 have Q(i $+ $#0)
    using Int_ZF_2_L1A zadd_int0 by simp
  moreover from T1 A3 have
    ∀k∈nat. Q(i $+ ($# k)) ⟶ Q(i $+ ($# succ(k)))
    by (rule Int_ZF_3_L6)
  ultimately have Q(i $+ ($# n)) by (rule ind_on_nat)
  with D2 show Q(k) by simp
qed
```

Induction on integer, implication between two forms of the induction step.

```
lemma (in int0) Int_ZF_3_L7A: assumes
  A1: ∀m. i≤m ∧ Q(m) ⟶ Q(m+1)
  shows ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
proof -
  { fix m assume i≤m ∧ Q(m)
    with A1 have T1: m∈ℤ Q(m+1) using Int_ZF_2_L1A by auto
    then have m+1 = m+($# 1) using Int_ZF_1_L8 by simp
    with T1 have Q(m $+ ($# 1)) using Int_ZF_1_L2
      by simp
  } then show thesis by simp
qed
```

Induction on integers, version with ZF increment function.

```
theorem (in int0) Induction_on_int:
  assumes A1: i≤k and A2: Q(i)
  and A3: ∀m. i≤m ∧ Q(m) ⟶ Q(m+1)
  shows Q(k)
proof -
  from A3 have ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
    by (rule Int_ZF_3_L7A)
  with A1 A2 show thesis by (rule Int_ZF_3_L7)
qed
```

Another form of induction on integers. This rewrites the basic theorem Int_ZF_3_L7 substituting $P(-k)$ for $Q(k)$.

```
lemma (in int0) Int_ZF_3_L7B: assumes A1: i≤k and A2: P($-i)
  and A3: ∀m. i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  shows P($-k)
proof -
  from A1 A2 A3 show P($-k) by (rule Int_ZF_3_L7)
qed
```

Another induction on integers. This rewrites Int_ZF_3_L7 substituting $-k$ for k and $-i$ for i.

```
lemma (in int0) Int_ZF_3_L8: assumes A1: k≤i and A2: P(i)
```

```
  and A3: ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  shows P(k)
proof -
  from A1 have T1: $-i≤$-k using Int_ZF_2_L10 by simp
  from A1 A2 have T2: P($- $- i) using Int_ZF_2_L1A zminus_zminus
    by simp
  from T1 T2 A3 have P($-($-k)) by (rule Int_ZF_3_L7)
  with A1 show P(k) using Int_ZF_2_L1A zminus_zminus by simp
qed
```

An implication between two forms of induction steps.

```
lemma (in int0) Int_ZF_3_L9: assumes A1: i∈ℤ
  and A2: ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
  shows ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
proof
  fix m show $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  proof
    assume A3: $- i ≤ m ∧ P($- m)
    then have $- i ≤ m by simp
    then have $-m ≤ $- ($- i) by (rule Int_ZF_2_L10)
    with A1 A2 A3 show P($-(m $+ ($# 1)))
      using zminus_zminus zminus_zadd_distrib by simp
  qed
qed
```

Backwards induction on integers, version with higher-order decrement function.

```
lemma (in int0) Int_ZF_3_L9A: assumes A1: k≤i and A2: P(i)
  and A3: ∀n. n≤i ∧ P(n) ⟶P(n $+ $-($#1))
  shows P(k)
proof -
  from A1 have T1: i∈ℤ using Int_ZF_2_L1A by simp
  from T1 A3 have T2: ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
    by (rule Int_ZF_3_L9)
  from A1 A2 T2 show P(k) by (rule Int_ZF_3_L8)
qed
```

Induction on integers, implication between two forms of the induction step.

```
lemma (in int0) Int_ZF_3_L10: assumes
  A1: ∀n. n≤i ∧ P(n) ⟶ P(n-1)
  shows ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
proof -
  { fix n assume n≤i ∧ P(n)
    with A1 have T1: n∈ℤ P(n-1) using Int_ZF_2_L1A by auto
    then have n-1 = n-($# 1) using Int_ZF_1_L8 by simp
    with T1 have P(n $+ $-($#1)) using Int_ZF_1_L10 by simp
  } then show thesis by simp
qed
```

Backwards induction on integers.

```
theorem (in int0) Back_induct_on_int:
  assumes A1: k≤i and A2: P(i)
  and A3: ∀n. n≤i ∧ P(n) ⟶ P(n-1)
  shows P(k)
proof -
  from A3 have ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
    by (rule Int_ZF_3_L10)
  with A1 A2 show P(k) by (rule Int_ZF_3_L9A)
qed
```

55.4 Bounded vs. finite subsets of integers

The goal of this section is to establish that a subset of integers is bounded is and only is it is finite. The fact that all finite sets are bounded is already shown for all linearly ordered groups in `OrderedGroups_ZF.thy`. To show the other implication we show that all intervals starting at 0 are finite and then use a result from `OrderedGroups_ZF.thy`.

There are no integers between k and $k + 1$.

```
lemma (in int0) Int_ZF_4_L1:
  assumes A1: k∈ℤ m∈ℤ n∈nat and A2: k $+ $#1 = m $+ $#n
  shows m =  k $+ $#1 ∨ m ≤ k
proof -
  { assume n=0
    with A1 A2 have m =  k $+ $#1 ∨ m ≤ k
      using zadd_int0 by simp }
  moreover
  { assume n≠0
    with A1 obtain j where D1: j∈nat n = succ(j)
      using Nat_ZF_1_L3 by auto
    with A1 A2 D1 have m =  k $+ $#1 ∨ m ≤ k
      using Int_ZF_2_L13 by simp }
  ultimately show thesis by blast
qed
```

A trivial calculation lemma that allows to subtract and add one.

```
lemma Int_ZF_4_L1A:
  assumes m∈int shows m $- $#1 $+ $#1 = m
  using assms eq_zdiff_iff by auto
```

There are no integers between k and $k + 1$, another formulation.

```
lemma (in int0) Int_ZF_4_L1B: assumes A1: m ≤ L
  shows
  m = L ∨ m+1 ≤ L
  m = L ∨ m ≤ L-1
proof -
```

```
  let k = L $- $#1
  from A1 have T1: m∈ℤ  L∈ℤ  L = k $+ $#1
    using Int_ZF_2_L1A Int_ZF_4_L1A by auto
  moreover from A1 obtain n where D1: n∈nat  L = m $+ $# n
    using Int_ZF_3_L2 by auto
  ultimately have m = L ∨ m ≤ k
    using Int_ZF_4_L1 by simp
  with T1 show m = L   ∨  m+1 ≤ L
    using Int_ZF_2_L9A by auto
  with T1 show m = L ∨ m ≤ L-1
    using Int_ZF_1_L8A Int_ZF_2_L9B by simp
qed
```

If $j \in m..k+1$, then $j \in m..n$ or $j = k+1$.

```
lemma (in int0) Int_ZF_4_L2: assumes A1: k∈ℤ
  and A2: j ∈ m..(k $+ $#1)
  shows j ∈ m..k ∨ j ∈ {k $+ $#1}
proof -
  from A2 have T1: m≤j j≤(k $+ $#1) using Order_ZF_2_L1A
    by auto
  then have T2: m∈ℤ j∈ℤ using Int_ZF_2_L1A by auto
  from T1 obtain n where n∈nat k $+ $#1 = j $+ $# n
    using Int_ZF_3_L2 by auto
  with A1 T1 T2 have (m≤j ∧ j ≤ k) ∨ j ∈ {k $+ $#1}
    using Int_ZF_4_L1 by auto
  then show thesis using Order_ZF_2_L1B by auto
qed
```

Extending an integer interval by one is the same as adding the new endpoint.

```
lemma (in int0) Int_ZF_4_L3: assumes A1: m≤ k
  shows m..(k $+ $#1) = m..k ∪ {k $+ $#1}
proof
  from A1 have T1: m∈ℤ k∈ℤ using Int_ZF_2_L1A by auto
  then show m .. (k $+ $# 1) ⊆ m .. k ∪ {k $+ $# 1}
    using Int_ZF_4_L2 by auto
  from T1 have m≤ m using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L3
    by simp
  with T1 A1 have m .. k ⊆ m .. (k $+ $# 1)
    using Int_ZF_2_L12 Int_ZF_2_L6 Order_ZF_2_L3 by simp
  with T1 A1 show m..k ∪ {k $+ $#1} ⊆ m..(k $+ $#1)
    using Int_ZF_2_L12A int_ord_is_refl Order_ZF_2_L2 by auto
qed
```

Integer intervals are finite - induction step.

```
lemma (in int0) Int_ZF_4_L4:
  assumes A1: i≤m and A2: i..m ∈ Fin(ℤ)
  shows i..(m $+ $#1) ∈ Fin(ℤ)
  using assms Int_ZF_4_L3 by simp
```

Integer intervals are finite.

```
lemma (in int0) Int_ZF_4_L5: assumes A1: i∈ℤ k∈ℤ
  shows i..k ∈ Fin(ℤ)
proof -
  { assume A2: i≤k
    moreover from A1 have i..i ∈ Fin(ℤ)
      using int_ord_is_refl Int_ZF_2_L4 Order_ZF_2_L4 by simp
    moreover from A2 have
      ∀m. i≤m ∧ i..m ∈ Fin(ℤ) ⟶ i..(m $+ $#1) ∈ Fin(ℤ)
      using Int_ZF_4_L4 by simp
    ultimately have i..k ∈ Fin(ℤ) by (rule Int_ZF_3_L7) }
  moreover
  { assume ¬ i ≤ k
    then have i..k ∈ Fin(ℤ) using Int_ZF_2_L6 Order_ZF_2_L5
      by simp }
  ultimately show thesis by blast
qed
```

Bounded integer sets are finite.

```
lemma (in int0) Int_ZF_4_L6: assumes A1: IsBounded(A,IntegerOrder)
  shows A ∈ Fin(ℤ)
proof -
  have T1: ∀m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder).
    $#0..m ∈ Fin(ℤ)
  proof
    fix m assume m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder)
    then have m∈ℤ using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L4E
      by auto
    then show $#0..m ∈ Fin(ℤ) using Int_ZF_4_L5 by simp
  qed
  have group3(ℤ,IntegerAddition,IntegerOrder)
    using Int_ZF_2_T1 by simp
  moreover from T1 have ∀m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder).
    Interval(IntegerOrder,TheNeutralElement(ℤ,IntegerAddition),m)
    ∈ Fin(ℤ) using Int_ZF_1_L8 by simp
  moreover note A1
  ultimately show A ∈ Fin(ℤ) by (rule group3.OrderedGroup_ZF_2_T1)
qed
```

A subset of integers is bounded iff it is finite.

```
theorem (in int0) Int_bounded_iff_fin:
  shows IsBounded(A,IntegerOrder)⟷ A∈Fin(ℤ)
  using Int_ZF_4_L6 Int_ZF_2_T1 group3.ord_group_fin_bounded
  by blast
```

The image of an interval by any integer function is finite, hence bounded.

```
lemma (in int0) Int_ZF_4_L8:
  assumes A1: i∈ℤ  k∈ℤ and A2: f:ℤ→ℤ
```

```
  shows
  f(i..k) ∈ Fin(ℤ)
  IsBounded(f(i..k),IntegerOrder)
  using assms Int_ZF_4_L5 Finite1_L6A Int_bounded_iff_fin
  by auto
```

If for every integer we can find one in A that is greater or equal, then A is is not bounded above, hence infinite.

```
lemma (in int0) Int_ZF_4_L9: assumes A1: ∀m∈ℤ. ∃k∈A. m≤k
  shows
  ¬IsBoundedAbove(A,IntegerOrder)
  A ∉ Fin(ℤ)
proof -
  have ℤ ≠ {0}
    using Int_ZF_1_L8A int_zero_not_one by blast
  with A1 show
    ¬IsBoundedAbove(A,IntegerOrder)
    A ∉ Fin(ℤ)
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L2A
    by auto
qed
```

55.5 Addition on integers in terms of magnitudes

In standard (informal) mathematics natural numbers form a subset of integers. In ZF that is not true as integers are defined as certain classes of pairs of natural numbers. As a result, addition on natural numbers is not a special case of addition on integers. The standard Isabelle/ZF's `Int` theory defines the notion of `zmagnitude` i.e. the magnitude of an integer. If z is an integer then `zmagnitude(z)` is its absolute value, interpreted as a natural number. The goal of this section is to provide facts about `zmagnitude` that are missing from the standard Isabelle/ZF's `Int` library and formulae that express addition of integers in terms of addition of their magnitudes.

The next lemma shows that `zmagnitude` of an integer is the same as `zmagnitude` of its opposite.

```
lemma (in int0) zmag_opposite_same: assumes z∈ℤ
  shows
    zmagnitude(z) = zmagnitude($-z)
    zmagnitude(z) = zmagnitude(-z)
proof -
  from assms obtain n where n∈nat and z=$#n ∨ z=$-($# succ(n))
    using int_cases by auto
  then show zmagnitude(z) = zmagnitude($-z) using zmagnitude_int_of
    by auto
  with assms show zmagnitude(z) = zmagnitude(-z) using Int_ZF_1_L9A
    by simp
```

qed

The magnitude of zero (the integer) is zero (the natural number) and the magnitude of one (the integer) is one (the natural number).

```
lemma (in int0) zmag_zero_one: shows zmagnitude(0) = 0 and zmagnitude(1)
= 1
proof -
  have zmagnitude(0) = zmagnitude($#0) and zmagnitude(1) = zmagnitude($#1)

    using Int_ZF_1_L8 by simp_all
  then show zmagnitude(0) = 0 and zmagnitude(1) = 1 using zmagnitude_int_of
    by simp_all
qed
```

If z_1, z_1 is a pair of integers then (at least) one of the following six cases holds:

1. Both integers are nonnegative.
2. Both integers are negative.
3. z_1 is nonnegative, z_2 is negative and magnitude of z_2 is less or equal than magnitude of z_1.
4. z_1 is nonnegative, z_2 is negative and magnitude of z_1 is (strictly) smaller than magnitude of z_2.
5. z_1 is negative, z_2 is nonnegative and magnitude of z_1 is less or equal than magnitude of z_2.
6. z_1 is negative, z_2 is nonnegative and magnitude of z_2 is (strictly) less than magnitude of z_1.

```
lemma (in int0) int_pair_6cases: assumes z₁∈ℤ z₂∈ℤ
  shows (0≤z₁ ∧ 0≤z₂) ∨ (z₁<0 ∧ z₂<0) ∨
    (0≤z₁ ∧ z₂<0 ∧ zmagnitude(z₂) ≤ zmagnitude(z₁)) ∨
    (0≤z₁ ∧ z₂<0 ∧ zmagnitude(z₁) < zmagnitude(z₂)) ∨
    (z₁<0 ∧ 0≤z₂ ∧ zmagnitude(z₁) ≤ zmagnitude(z₂)) ∨
    (z₁<0 ∧ 0≤z₂ ∧ zmagnitude(z₂) < zmagnitude(z₁))
  using assms int_nonneg_or_neg zmagnitude_type nat_order_2cases by blast
```

Sum of nonnegative integers in nonnegative. The magnitude of the sum of such integers is the sum of their magnitudes. We can add nonnegative integers by adding their magnitudes and converting the result to an integer. `zmagnitude` (defined in standard Isabelle/ZF's `Int` theory) is the natural number corresponding to the absolute value of an integer number.

```
lemma (in int0) add_nonneg_ints: assumes 0≤z₁ 0≤z₂
  shows
    0≤z₁+z₂
    zmagnitude(z₁+z₂) = zmagnitude(z₁) #+ zmagnitude(z₂)
    z₁+z₂ = $# (zmagnitude(z₁) #+ zmagnitude(z₂))
proof -
```

```
  let m₁ = zmagnitude(z₁)
  let m₂ = zmagnitude(z₂)
  from assms show 0≤z₁+z₂ using OrderedGroup_ZF_1_L12 by simp
  from assms show z₁+z₂ = $#(m₁ #+ m₂)
    using nonnegative_not_znegative int_of_add Int_ZF_1_L2(1) by simp
  then show zmagnitude(z₁+z₂) = m₁ #+ m₂
    using add_type natify_ident zmagnitude_int_of by simp
qed
```

Sum of negative integers is negative. The magnitude of the sum of such integers is the sum of their magnitudes. We can calculate the sum of negative integers by taking the sum of their magnitudes, converting that to an integer and taking negative of the result.

```
lemma (in int0) add_neg_ints: assumes z₁<0 z₂<0
  shows
    z₁+z₂<0
    zmagnitude(z₁+z₂) = zmagnitude(z₁) #+ zmagnitude(z₂)
    z₁+z₂ = $-($#(zmagnitude(z₁) #+ zmagnitude(z₂)))
proof -
  from assms show z₁+z₂<0
    using Int_ZF_2_T1(2,3) group3.group_less_less by simp
  let m₁ = zmagnitude(z₁)
  let m₂ = zmagnitude(z₂)
  from assms show z₁+z₂ = $-($#(m₁ #+ m₂))
    using znegative_as_ls_zero zneg_mag int_of_add zminus_zadd_distrib

      zadd_type zminus_zminus Int_ZF_1_L2(1) less_are_members by auto
  then show zmagnitude(z₁+z₂) = m₁ #+ m₂
    using add_type zmagnitude_zminus_int_of by simp
qed
```

If z_1 is a nonnegative integer and z_2 is a negative integer with a less or equal magnitude, then their sum is nonnegative and its magnitude is the difference between magnitudes of z_1 and z_2.

```
lemma (in int0) add_nonneg_neg1:
  assumes 0≤z₁  z₂<0 zmagnitude(z₂) ≤ zmagnitude(z₁)
  shows
    0≤z₁+z₂
    zmagnitude(z₁+z₂) = zmagnitude(z₁) #- zmagnitude(z₂)
    z₁+z₂ = $#(zmagnitude(z₁) #- zmagnitude(z₂))
proof -
  let m₁ = zmagnitude(z₁)
  let m₂ = zmagnitude(z₂)
  from assms(1,2) have
    z₁∈ℤ z₁ = $#m₁ and z₂∈ℤ z₂ = $-($#(m₂))
    using nonnegative_not_znegative not_zneg_mag
      znegative_as_ls_zero less_are_members zneg_mag zminus_zminus
    by auto
```

```
  with assms(3) show  z₁+z₂ = $#(m₁ #- m₂)
    using Int_ZF_1_L2(1) zmagnitude_type int_of_diff unfolding zdiff_def

    by simp
  then show zmagnitude(z₁+z₂) = m₁ #- m₂
    using zmagnitude_type diff_type zmagnitude_int_of by simp
  from ‹z₁+z₂ = $#(m₁ #- m₂)› ‹z₁∈ℤ› ‹z₂∈ℤ› show 0≤z₁+z₂
    using not_znegative_int_of notzneg_is_geq_zero Int_ZF_1_T2(3) group0.group_op_closed

    by simp
qed
```

If z_1 is a nonnegative integer and z_2 is a negative integer with a greater magnitude, then their sum is negative and its magnitude is the difference between magnitudes of z_2 and z_1.

```
lemma (in int0) add_nonneg_neg2:
  assumes 0≤z₁  z₂<0  zmagnitude(z₁) < zmagnitude(z₂)
  shows
    z₁+z₂ < 0
    zmagnitude(z₁+z₂) = zmagnitude(z₂) #- zmagnitude(z₁)
    z₁+z₂ = $-($#(zmagnitude(z₂) #- zmagnitude(z₁)))
proof -
  let m₁ = zmagnitude(z₁)
  let m₂ = zmagnitude(z₂)
  from assms(1,2) have
    z₁∈ℤ ¬znegative(z₁) z₁ = $#m₁ and
    z₂∈ℤ znegative(z₂) z₂ = $-($#(m₂))
    using nonnegative_not_znegative not_zneg_mag less_are_members
      znegative_as_ls_zero zneg_mag zminus_zminus by auto
  then have z₁+z₂ = ($#m₁) $- ($#(m₂))
    using Int_ZF_1_L2(1) unfolding zdiff_def by simp
  with assms(3) show z₁+z₂ < 0
    using zmagnitude_type zless_int_of zdiff_type znegative_as_ls_zero
    unfolding zless_def by simp
  from assms(3) ‹z₁+z₂ = ($#m₁) $- ($#(m₂))›
  show z₁+z₂ = $-($#(m₂ #- m₁))
    using leI zmagnitude_type int_of_diff zminus_zdiff_eq by simp
  then show zmagnitude(z₁+z₂) = m₂ #- m₁
    using zmagnitude_zminus_int_of by simp
qed
```

If z_1 is a negative integer and z_2 is a nonnegative integer with a greater or equal magnitude, then their sum is nonnegative and its magnitude is the difference between magnitudes of z_2 and z_1. This is essentially `add_nonneg_neg1` with z_1 and z_2 swapped.

```
lemma (in int0) add_neg_nonneg1:
  assumes z₁<0 0≤z₂ zmagnitude(z₁) ≤ zmagnitude(z₂)
  shows
```

```
  0≤z₁+z₂
  zmagnitude(z₁+z₂) = zmagnitude(z₂) #- zmagnitude(z₁)
  z₁+z₂ = $#(zmagnitude(z₂) #- zmagnitude(z₁))
proof -
  from assms(1,2) have z₁+z₂ = z₂+z₁
    using OrderedGroup_ZF_1_L4(2) less_are_members(1) Int_ZF_1_L4(1)
    by auto
  with assms show
      0≤z₁+z₂
      zmagnitude(z₁+z₂) = zmagnitude(z₂) #- zmagnitude(z₁)
      z₁+z₂ = $#(zmagnitude(z₂) #- zmagnitude(z₁))
    using add_nonneg_neg1 by simp_all
qed
```

If z_1 is a negative integer and z_2 is a nonnegative integer with a smaller magnitude, the their sum is negative and its magnitude is the difference between magnitudes of z_1 and z_2. This is essentially `add_nonneg_neg2` with z_1 and z_2 swapped.

```
lemma (in int0) add_neg_nonneg2:
  assumes z₁<0 0≤z₂ zmagnitude(z₂) < zmagnitude(z₁)
  shows
    z₁+z₂ < 0
    zmagnitude(z₁+z₂) = zmagnitude(z₁) #- zmagnitude(z₂)
    z₁+z₂ = $-($#(zmagnitude(z₁) #- zmagnitude(z₂)))
proof -
  from assms(1,2) have z₁+z₂ = z₂+z₁
    using OrderedGroup_ZF_1_L4(2) less_are_members(1) Int_ZF_1_L4(1)
    by auto
  with assms show
    z₁+z₂ < 0
    zmagnitude(z₁+z₂) = zmagnitude(z₁) #- zmagnitude(z₂)
    z₁+z₂ = $-($#(zmagnitude(z₁) #- zmagnitude(z₂)))
    using add_nonneg_neg2 by simp_all
qed
```

`zmagnitud` respects multiplication

```
lemma (in int0) zmagnitud_mult:
  assumes x∈ℤ y∈ℤ
  shows zmagnitude(x·y) = zmagnitude(x) #* zmagnitude(y)
proof-
  from assms(1) int_cases obtain n where n:n∈nat x=$#n ∨ x=$- $# succ(n)
unfolding ints_def by blast
  from assms(2) int_cases obtain m where m:m∈nat y=$#m ∨ y=$- $# succ(m)
unfolding ints_def by blast
  {
    fix q t assume as:q∈nat t∈nat
    have ($# q) $* ($# t) = (intrel{<q,0>}) $* (intrel{<t,0>})
      unfolding int_of_def using as by auto
```

```
    then have ($# q) $* ($# t) = (intrel{<(q #* t) #+ (0 #* 0),(q #*
0) #+ (0 #* t)>})
      using zmult as(1) nat_0I as(2) nat_0I by auto
    then have ($# q) $* ($# t) = intrel{<(q #*t),0>} using mult_0 mult_0_right
      add_0_right by auto
    then have ($# q) $* ($# t) = $#(q #* t) unfolding int_of_def by auto
  }
  then have R:⋀q t. q∈nat ⟹ t∈nat ⟹ ($# q) $* ($# t) = $#(q #* t)
by auto
  {
    assume as:x=$#n y=$#m
    from R n(1) m(1) have ($# n) $* ($# m) = $# (n #* m) by auto
    then have zmagnitude(($# n) $* ($# m)) = zmagnitude($#(n #* m)) by
auto
    then have zmagnitude(x $* y) = n #* m using zmagnitude_int_of as
by auto
    then have zmagnitude(x $* y) = zmagnitude(x) #* zmagnitude(y)
      using as n(1) m(1) by auto
    then have zmagnitude(x·y) = zmagnitude(x) #* zmagnitude(y)
      using Int_ZF_1_L2(2) assms by auto
  } moreover
  {
    assume as:x=$#n y=$- $#succ(m)
    have A:($# n) $* ($- $#succ(m)) = $- ($# n $* $#succ(m))
      using zmult_zminus_right by auto
    moreover have succ(m)∈nat using nat_succI m(1) by auto
    ultimately have $# n $* $#succ(m) = $#(n #* succ(m))
      using R n(1) by auto
    with A have zmagnitude(($# n) $* ($- $#succ(m))) = zmagnitude($-
$#(n #* succ(m)))
      by auto
    with as have zmagnitude(x $* y) = n #* succ(m)
      using zmagnitude_zminus_int_of  by auto
    then have zmagnitude(x $* y) = zmagnitude(x) #* zmagnitude(y)
      using as n(1) m(1) by auto
    then have zmagnitude(x·y) = zmagnitude(x) #* zmagnitude(y)
      using Int_ZF_1_L2(2) assms by auto
  } moreover
  {
    assume as:x=$- $#succ(n) y=$#m
    have A:($- $#succ(n)) $* ($#m) = $- ($# succ(n) $* $#m)
      using zmult_zminus by auto
    moreover have succ(n)∈nat using nat_succI n(1) by auto
    ultimately have $# succ(n) $* $#m = $#(succ(n) #* m)
      using R m(1) by auto
    with A have zmagnitude(($- $# succ(n)) $* $#m) = zmagnitude($- $#(succ(n)
#* m))
      by auto
    with as have zmagnitude(x $* y) = succ(n) #* m
```

```
      using zmagnitude_zminus_int_of by auto
    then have zmagnitude(x $* y) = zmagnitude(x) #* zmagnitude(y)
      using as n(1) m(1) by auto
    then have zmagnitude(x·y) = zmagnitude(x) #* zmagnitude(y)
      using Int_ZF_1_L2(2) assms by auto
  }moreover
  {
    assume as:x=$- $#succ(n) y=$- $#succ(m)
    have A:($- $#succ(n)) $* ($- $#succ(m)) = ($# succ(n) $* $#succ(m))
      using zmult_zminus by auto
    moreover have succ(n)∈nat succ(m)∈nat using nat_succI n(1) m(1)
by auto
    ultimately have $# succ(n) $* $#succ(m) = $#(succ(n) #* succ(m))
      using R by auto
    with A have zmagnitude(($- $# succ(n)) $* $- $#succ(m)) = zmagnitude($#(succ(n)
#*succ(m)))
      by auto
    with as have zmagnitude(x $* y) = succ(n) #* succ(m)
      using zmagnitude_zminus_int_of by auto
    then have zmagnitude(x $* y) = zmagnitude(x) #* zmagnitude(y)
      using as n(1) m(1) by auto
    then have zmagnitude(x·y) = zmagnitude(x) #* zmagnitude(y)
      using Int_ZF_1_L2(2) assms by auto
  } ultimately
  show thesis using n(2) m(2) by auto
qed

end
```

56 Integers 1

```
theory Int_ZF_1 imports Int_ZF_IML OrderedRing_ZF

begin
```

This theory file considers the set of integers as an ordered ring.

56.1 Integers as a ring

In this section we show that integers form a commutative ring.

The next lemma provides the condition to show that addition is distributive with respect to multiplication.

```
lemma (in int0) Int_ZF_1_1_L1: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a·(b+c) = a·b + a·c
  (b+c)·a = b·a + c·a
  using assms Int_ZF_1_L2 zadd_zmult_distrib zadd_zmult_distrib2
```

```
  by auto
```

Integers form a commutative ring, hence we can use theorems proven in ring0 context (locale).

```
lemma (in int0) Int_ZF_1_1_L2: shows
  IsAring(ℤ,IntegerAddition,IntegerMultiplication)
  IntegerMultiplication {is commutative on} ℤ
  ring0(ℤ,IntegerAddition,IntegerMultiplication)
proof -
  have ∀a∈ℤ.∀b∈ℤ.∀c∈ℤ.
    a·(b+c) = a·b + a·c ∧ (b+c)·a = b·a + c·a
    using Int_ZF_1_1_L1 by simp
  then have IsDistributive(ℤ,IntegerAddition,IntegerMultiplication)
    using IsDistributive_def by simp
  then show IsAring(ℤ,IntegerAddition,IntegerMultiplication)
    ring0(ℤ,IntegerAddition,IntegerMultiplication)
    using Int_ZF_1_T1 Int_ZF_1_T2 IsAring_def ring0_def
    by auto
  have ∀a∈ℤ.∀b∈ℤ. a·b = b·a using Int_ZF_1_L4 by simp
  then show IntegerMultiplication {is commutative on} ℤ
    using IsCommutative_def by simp
qed
```

Zero and one are integers.

```
lemma (in int0) int_zero_one_are_int: shows 0∈ℤ  1∈ℤ
  using Int_ZF_1_1_L2 ring0.Ring_ZF_1_L2 by auto
```

Negative of zero is zero.

```
lemma (in int0) int_zero_one_are_intA: shows (-0) = 0
  using Int_ZF_1_T2 group0.group_inv_of_one by simp
```

Properties with one integer.

```
lemma (in int0) Int_ZF_1_1_L4: assumes A1: a ∈ ℤ
  shows
  a+0 = a
  0+a = a
  a·1 = a   1·a = a
  0·a = 0   a·0 = 0
  (-a) ∈ ℤ  (-(-a)) = a
  a-a = 0   a-0 = a  2·a = a+a
proof -
  from A1 show
    a+0 = a   0+a = a   a·1 = a
    1·a = a   a-a = 0   a-0 = a
    (-a) ∈ ℤ  2·a = a+a   (-(-a)) = a
    using Int_ZF_1_1_L2 ring0.Ring_ZF_1_L3 by auto
  from A1 show 0·a = 0   a·0 = 0
    using Int_ZF_1_1_L2 ring0.Ring_ZF_1_L6 by auto
```

qed

Properties that require two integers.

```
lemma (in int0) Int_ZF_1_1_L5: assumes a∈ℤ  b∈ℤ
  shows
  a+b ∈ ℤ
  a-b ∈ ℤ
  a·b ∈ ℤ
  a+b = b+a
  a·b = b·a
  (-b)-a = (-a)-b
  (-(a+b)) = (-a)-b
  (-(a-b)) = ((-a)+b)
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  (-a)·(-b) = a·b
  using assms Int_ZF_1_1_L2 ring0.Ring_ZF_1_L4 ring0.Ring_ZF_1_L9
    ring0.Ring_ZF_1_L7 ring0.Ring_ZF_1_L7A Int_ZF_1_L4 by auto
```

2 and 3 are integers.

```
lemma (in int0) int_two_three_are_int: shows 2 ∈ ℤ  3 ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 by auto
```

Another property with two integers.

```
lemma (in int0) Int_ZF_1_1_L5B:
  assumes a∈ℤ  b∈ℤ
  shows a-(-b) = a+b
  using assms Int_ZF_1_1_L2 ring0.Ring_ZF_1_L9
  by simp
```

Properties that require three integers.

```
lemma (in int0) Int_ZF_1_1_L6: assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a-(b+c) = a-b-c
  a-(b-c) = a-b+c
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using assms Int_ZF_1_1_L2 ring0.Ring_ZF_1_L10  ring0.Ring_ZF_1_L8
  by auto
```

One more property with three integers.

```
lemma (in int0) Int_ZF_1_1_L6A: assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows a+(b-c) = a+b-c
  using assms Int_ZF_1_1_L2 ring0.Ring_ZF_1_L10A by simp
```

Associativity of addition and multiplication.

```
lemma (in int0) Int_ZF_1_1_L7: assumes a∈ℤ b∈ℤ c∈ℤ
  shows
```

```
  a+b+c = a+(b+c)
  a·b·c = a·(b·c)
  using assms Int_ZF_1_1_L2 ring0.Ring_ZF_1_L11 by auto
```

56.2 Rearrangement lemmas

In this section we collect lemmas about identities related to rearranging the terms in expresssions

A formula with a positive integer.

```
lemma (in int0) Int_ZF_1_2_L1: assumes 0≤a
  shows abs(a)+1 = abs(a+1)
  using assms Int_ZF_2_L16 Int_ZF_2_L12A by simp
```

A formula with two integers, one positive.

```
lemma (in int0) Int_ZF_1_2_L2: assumes A1: a∈ℤ and A2: 0≤b
  shows a+(abs(b)+1)·a = (abs(b+1)+1)·a
proof -
  from A2 have abs(b+1) ∈ ℤ
    using Int_ZF_2_L12A Int_ZF_2_L1A Int_ZF_2_L14 by blast
  with A1 A2 show thesis
    using Int_ZF_1_2_L1 Int_ZF_1_1_L2 ring0.Ring_ZF_2_L1
    by simp
qed
```

A couple of formulae about canceling opposite integers.

```
lemma (in int0) Int_ZF_1_2_L3: assumes A1: a∈ℤ  b∈ℤ
  shows
  a+b-a = b
  a+(b-a) = b
  a+b-b = a
  a-b+b = a
  (-a)+(a+b) = b
  a+(b-a) = b
  (-b)+(a+b) = a
  a-(b+a) = -b
  a-(a+b) = -b
  a-(a-b) = b
  a-b-a =  -b
  a-b - (a+b) = (-b)-b
  using assms Int_ZF_1_T2 group0.group0_4_L6A group0.inv_cancel_two
    group0.group0_2_L16A group0.group0_4_L6AA group0.group0_4_L6AB
    group0.group0_4_L6F group0.group0_4_L6AC by auto
```

Subtracting one does not increase integers. This may be moved to a theory about ordered rings one day.

```
lemma (in int0) Int_ZF_1_2_L3A: assumes A1: a≤b
  shows a-1 ≤ b
```

```
proof -
  from A1 have b+1-1 = b
    using Int_ZF_2_L1A int_zero_one_are_int Int_ZF_1_2_L3 by simp
  moreover from A1 have a-1 ≤ b+1-1
    using Int_ZF_2_L12A int_zero_one_are_int Int_ZF_1_1_L4 int_ord_transl_inv
    by simp
  ultimately show a-1 ≤ b by simp
qed
```

Subtracting one does not increase integers, special case.

```
lemma (in int0) Int_ZF_1_2_L3AA:
  assumes A1: a∈ℤ shows
  a-1 ≤a
  a-1 ≠ a
  ¬(a≤a-1)
  ¬(a+1 ≤a)
  ¬(1+a ≤a)
proof -
  from A1 have a≤a using int_ord_is_refl refl_def
    by simp
  then show a-1 ≤a using Int_ZF_1_2_L3A
    by simp
  moreover from A1 show a-1 ≠ a using Int_ZF_1_L14 by simp
  ultimately show I: ¬(a≤a-1) using Int_ZF_2_L19AA
    by blast
  with A1 show ¬(a+1 ≤a)
    using int_zero_one_are_int Int_ZF_2_L9B by simp
  with A1 show ¬(1+a ≤a)
    using int_zero_one_are_int Int_ZF_1_1_L5 by simp
qed
```

A formula with a nonpositive integer.

```
lemma (in int0) Int_ZF_1_2_L4: assumes a≤0
  shows abs(a)+1 = abs(a-1)
  using assms int_zero_one_are_int Int_ZF_1_2_L3A Int_ZF_2_T1
      group3.OrderedGroup_ZF_3_L3A Int_ZF_2_L1A
      int_zero_one_are_int Int_ZF_1_1_L5 by simp
```

A formula with two integers, one negative.

```
lemma (in int0) Int_ZF_1_2_L5: assumes A1: a∈ℤ and A2: b≤0
  shows a+(abs(b)+1)·a = (abs(b-1)+1)·a
proof -
  from A2 have abs(b-1) ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_2_L3A Int_ZF_2_L1A Int_ZF_2_L14

    by blast
  with A1 A2 show thesis
    using Int_ZF_1_2_L4 Int_ZF_1_1_L2 ring0.Ring_ZF_2_L1
    by simp
```

```
qed
```

A rearrangement with four integers.

```
lemma (in int0) Int_ZF_1_2_L6:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows
  a-(b-1)·c = (d-b·c)-(d-a-c)
proof -
  from A1 have T1:
    (d-b·c) ∈ ℤ d-a ∈ ℤ (-(b·c)) ∈ ℤ
    using Int_ZF_1_1_L5 Int_ZF_1_1_L4 by auto
  with A1 have
    (d-b·c)-(d-a-c) = (-(b·c))+a+c
    using Int_ZF_1_1_L6 Int_ZF_1_2_L3 by simp
  also from A1 T1 have (-(b·c))+a+c = a-(b-1)·c
    using int_zero_one_are_int Int_ZF_1_1_L6 Int_ZF_1_1_L4 Int_ZF_1_1_L5
    by simp
  finally show thesis by simp
qed
```

Some other rearrangements with two integers.

```
lemma (in int0) Int_ZF_1_2_L7: assumes a∈ℤ  b∈ℤ
  shows
  a·b = (a-1)·b+b
  a·(b+1) = a·b+a
  (b+1)·a = b·a+a
  (b+1)·a = a+b·a
  using assms Int_ZF_1_1_L1 Int_ZF_1_1_L5 int_zero_one_are_int
    Int_ZF_1_1_L6 Int_ZF_1_1_L4 Int_ZF_1_T2 group0.inv_cancel_two
  by auto
```

Another rearrangement with two integers.

```
lemma (in int0) Int_ZF_1_2_L8:
  assumes A1: a∈ℤ b∈ℤ
  shows a+1+(b+1) = b+a+2
  using assms int_zero_one_are_int Int_ZF_1_T2 group0.group0_4_L8
  by simp
```

A couple of rearrangement with three integers.

```
lemma (in int0) Int_ZF_1_2_L9:
  assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  (a-b)+(b-c) = a-c
  (a-b)-(a-c) = c-b
  a+(b+(c-a-b)) = c
  (-a)-b+c = c-a-b
  (-b)-a+c = c-a-b
  (-((-a)+b+c)) = a-b-c
```

```
  a+b+c-a = b+c
  a+b-(a+c) = b-c
  using assms Int_ZF_1_T2
    group0.group0_4_L4B group0.group0_4_L6D group0.group0_4_L4D
    group0.group0_4_L6B group0.group0_4_L6E
  by auto
```

Another couple of rearrangements with three integers.

```
lemma (in int0) Int_ZF_1_2_L9A:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows (-(a-b-c)) = c+b-a
proof -
  from A1 have T:
    a-b ∈ ℤ  (-(a-b)) ∈ ℤ  (-b) ∈ ℤ using
    Int_ZF_1_1_L4 Int_ZF_1_1_L5 by auto
  with A1 have (-(a-b-c)) = c - ((-b)+a)
     using Int_ZF_1_1_L5 by simp
   also from A1 T have ... = c+b-a
     using Int_ZF_1_1_L6 Int_ZF_1_1_L5B
     by simp
  finally show (-(a-b-c)) = c+b-a
    by simp
qed
```

Another rearrangement with three integers.

```
lemma (in int0) Int_ZF_1_2_L10:
  assumes A1: a∈ℤ b∈ℤ c∈ℤ
  shows (a+1)·b + (c+1)·b = (c+a+2)·b
proof -
  from A1 have a+1 ∈ ℤ c+1 ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 by auto
  with A1 have
    (a+1)·b + (c+1)·b = (a+1+(c+1))·b
    using Int_ZF_1_1_L1 by simp
  also from A1 have ... = (c+a+2)·b
    using Int_ZF_1_2_L8 by simp
  finally show thesis by simp
qed
```

A technical rearrangement involing inequalities with absolute value.

```
lemma (in int0) Int_ZF_1_2_L10A:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  e∈ℤ
  and A2: abs(a·b-c) ≤ d  abs(b·a-e) ≤ f
  shows abs(c-e) ≤  f+d
proof -
  from A1 A2 have T1:
    d∈ℤ  f∈ℤ  a·b ∈ ℤ  a·b-c ∈ ℤ  b·a-e ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  with A2 have
```

```
    abs((b·a-e)-(a·b-c)) ≤ f +d
    using Int_ZF_2_L21 by simp
  with A1 T1 show abs(c-e) ≤ f+d
    using Int_ZF_1_1_L5 Int_ZF_1_2_L9 by simp
qed
```

Some arithmetics.

```
lemma (in int0) Int_ZF_1_2_L11: assumes A1: a∈ℤ
  shows
  a+1+2 = a+3
  a = 2·a - a
proof -
  from A1 show a+1+2 = a+3
    using int_zero_one_are_int int_two_three_are_int Int_ZF_1_T2 group0.group0_4_L4C
    by simp
  from A1 show a = 2·a - a
    using int_zero_one_are_int Int_ZF_1_1_L1 Int_ZF_1_1_L4 Int_ZF_1_T2
group0.inv_cancel_two
    by simp
qed
```

A simple rearrangement with three integers.

```
lemma (in int0) Int_ZF_1_2_L12:
  assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  (b-c)·a = a·b - a·c
  using assms Int_ZF_1_1_L6 Int_ZF_1_1_L5 by simp
```

A big rearrangement with five integers.

```
lemma (in int0) Int_ZF_1_2_L13:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ  x∈ℤ
  shows (x+(a·x+b)+c)·d = d·(a+1)·x + (b·d+c·d)
proof -
  from A1 have T1:
    a·x ∈ ℤ   (a+1)·x ∈ ℤ
    (a+1)·x + b ∈ ℤ
    using Int_ZF_1_1_L5 int_zero_one_are_int by auto
  with A1 have (x+(a·x+b)+c)·d = ((a+1)·x + b)·d + c·d
    using Int_ZF_1_1_L7 Int_ZF_1_2_L7 Int_ZF_1_1_L1
    by simp
  also from A1 T1 have ... = (a+1)·x·d + b · d + c·d
    using Int_ZF_1_1_L1 by simp
  finally have (x+(a·x+b)+c)·d = (a+1)·x·d + b·d + c·d
    by simp
  moreover from A1 T1 have (a+1)·x·d = d·(a+1)·x
    using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_1_1_L7 by simp
  ultimately have (x+(a·x+b)+c)·d = d·(a+1)·x + b·d + c·d
    by simp
  moreover from A1 T1 have
```

```
    d·(a+1)·x ∈ ℤ  b·d ∈ ℤ  c·d ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 by auto
  ultimately show thesis using Int_ZF_1_1_L7 by simp
qed
```

Rerrangement about adding linear functions.

```
lemma (in int0) Int_ZF_1_2_L14:
  assumes a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ  x∈ℤ
  shows (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
  using assms Int_ZF_1_1_L2 ring0.Ring_ZF_2_L3 by simp
```

A rearrangement with four integers. Again we have to use the generic set notation to use a theorem proven in different context.

```
lemma (in int0) Int_ZF_1_2_L15: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ
  and A2: a = b-c-d
  shows
  d = b-a-c
  d = (-a)+b-c
  b = a+d+c
proof -
  let G = int
  let f = IntegerAddition
  from A1 A2 have I:
    group0(G, f)   f {is commutative on} G
    a ∈ G  b ∈ G c ∈ G  d ∈ G
    a = f⟨f⟨b,GroupInv(G, f)(c)⟩,GroupInv(G, f)(d)⟩
    using Int_ZF_1_T2 by auto
  then have
    d = f⟨f⟨b,GroupInv(G, f)(a)⟩,GroupInv(G,f)(c)⟩
    by (rule group0.group0_4_L9)
  then show d = b-a-c by simp
  from I have d = f⟨f⟨GroupInv(G, f)(a),b⟩, GroupInv(G, f)(c)⟩
    by (rule group0.group0_4_L9)
  thus d = (-a)+b-c
    by simp
  from I have b = f⟨f⟨a, d⟩,c⟩
    by (rule group0.group0_4_L9)
  thus b = a+d+c by simp
qed
```

A rearrangement with four integers. Property of groups.

```
lemma (in int0) Int_ZF_1_2_L16:
  assumes a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ
  shows a+(b-c)+d = a+b+d-c
  using assms Int_ZF_1_T2 group0.group0_4_L8 by simp
```

Some rearrangements with three integers. Properties of groups.

```
lemma (in int0) Int_ZF_1_2_L17:
```

```
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a+b-c+(c-b) = a
  a+(b+c)-c = a+b
proof -
  let G = int
  let f = IntegerAddition
  from A1 have I:
    group0(G, f)
    a ∈ G  b ∈ G c ∈ G
    using Int_ZF_1_T2 by auto
  then have
    f⟨f⟨f⟨a,b⟩,GroupInv(G, f)(c)⟩,f⟨c,GroupInv(G, f)(b)⟩⟩ = a
    by (rule group0.group0_2_L14A)
  thus a+b-c+(c-b) = a by simp
  from I have
    f⟨f⟨a,f⟨b,c⟩⟩,GroupInv(G, f)(c)⟩ = f⟨a,b⟩
    by (rule group0.group0_2_L14A)
  thus a+(b+c)-c = a+b by simp
qed
```

Another rearrangement with three integers. Property of abelian groups.

```
lemma (in int0) Int_ZF_1_2_L18:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows a+b-c+(c-a) = b
proof -
  let G = int
  let f = IntegerAddition
  from A1 have
    group0(G, f)   f {is commutative on} G
    a ∈ G  b ∈ G c ∈ G
    using Int_ZF_1_T2 by auto
  then have
    f⟨f⟨f⟨a,b⟩,GroupInv(G, f)(c)⟩,f⟨c,GroupInv(G, f)(a)⟩⟩ = b
    by (rule group0.group0_4_L6D)
  thus a+b-c+(c-a) = b by simp
qed
```

56.3 Integers as an ordered ring

We already know from `Int_ZF` that integers with addition form a linearly ordered group. To show that integers form an ordered ring we need the fact that the set of nonnegative integers is closed under multiplication.

We start with the property that a product of nonnegative integers is nonnegative. The proof is by induction and the next lemma is the induction step.

```
lemma (in int0) Int_ZF_1_3_L1: assumes A1: 0≤a  0≤b
```

```
  and A3: 0 ≤ a·b
  shows 0 ≤ a·(b+1)
proof -
  from A1 A3 have 0+0 ≤ a·b+a
    using int_ineq_add_sides by simp
  with A1 show 0 ≤ a·(b+1)
    using int_zero_one_are_int Int_ZF_1_1_L4 Int_ZF_2_L1A Int_ZF_1_2_L7

    by simp
qed
```

Product of nonnegative integers is nonnegative.

```
lemma (in int0) Int_ZF_1_3_L2: assumes A1: 0≤a  0≤b
  shows 0≤a·b
proof -
  from A1 have 0≤b by simp
  moreover from A1 have 0 ≤ a·0 using
    Int_ZF_2_L1A Int_ZF_1_1_L4 int_zero_one_are_int int_ord_is_refl refl_def
    by simp
  moreover from A1 have
    ∀m. 0≤m ∧ 0≤a·m ⟶ 0 ≤ a·(m+1)
    using Int_ZF_1_3_L1 by simp
  ultimately show 0≤a·b by (rule Induction_on_int)
qed
```

The set of nonnegative integers is closed under multiplication.

```
lemma (in int0) Int_ZF_1_3_L2A: shows
  ℤ⁺ {is closed under} IntegerMultiplication
proof -
  { fix a b assume a∈ℤ⁺  b∈ℤ⁺
    then have a·b ∈ℤ⁺
      using Int_ZF_1_3_L2 Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L2
      by simp
  } then have ∀a∈ℤ⁺.∀b∈ℤ⁺.a·b ∈ℤ⁺ by simp
  then show thesis using IsOpClosed_def by simp
qed
```

Integers form an ordered ring. All theorems proven in the `ring1` context are valid in `int0` context.

```
theorem (in int0) Int_ZF_1_3_T1: shows
  IsAnOrdRing(ℤ,IntegerAddition,IntegerMultiplication,IntegerOrder)
  ring1(ℤ,IntegerAddition,IntegerMultiplication,IntegerOrder)
  using Int_ZF_1_1_L2 Int_ZF_2_L1B Int_ZF_1_3_L2A Int_ZF_2_T1
    OrdRing_ZF_1_L6 OrdRing_ZF_1_L2 by auto
```

Product of integers that are greater that one is greater than one. The proof is by induction and the next step is the induction step.

```
lemma (in int0) Int_ZF_1_3_L3_indstep:
```

```
  assumes A1: 1≤a  1≤b
  and A2: 1 ≤ a·b
  shows 1 ≤ a·(b+1)
proof -
   from A1 A2 have 1≤2 and 2 ≤ a·(b+1)
    using Int_ZF_2_L1A int_ineq_add_sides Int_ZF_2_L16B Int_ZF_1_2_L7

    by auto
  then show 1 ≤ a·(b+1) by (rule Int_order_transitive)
qed
```

Product of integers that are greater that one is greater than one.

```
lemma (in int0) Int_ZF_1_3_L3:
  assumes A1: 1≤a 1≤b
  shows 1 ≤ a·b
proof -
  from A1 have 1≤b  1≤a·1
    using Int_ZF_2_L1A Int_ZF_1_1_L4 by auto
  moreover from A1 have
    ∀m. 1≤m ∧ 1 ≤ a·m ⟶ 1 ≤ a·(m+1)
    using Int_ZF_1_3_L3_indstep by simp
  ultimately show 1 ≤ a·b by (rule Induction_on_int)
qed
```

$|a \cdot (-b)| = |(-a) \cdot b| = |(-a) \cdot (-b)| = |a \cdot b|$ This is a property of ordered rings..

```
lemma (in int0) Int_ZF_1_3_L4: assumes a∈ℤ  b∈ℤ
  shows
  abs((-a)·b) = abs(a·b)
  abs(a·(-b)) = abs(a·b)
  abs((-a)·(-b)) = abs(a·b)
  using assms Int_ZF_1_1_L5 Int_ZF_2_L17 by auto
```

Absolute value of a product is the product of absolute values. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L5:
  assumes A1: a∈ℤ  b∈ℤ
  shows abs(a·b) = abs(a)·abs(b)
  using assms Int_ZF_1_3_T1 ring1.OrdRing_ZF_2_L5 by simp
```

Double nonnegative is nonnegative. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L5A: assumes 0≤a
  shows 0≤2·a
  using assms Int_ZF_1_3_T1 ring1.OrdRing_ZF_1_L5A by simp
```

The next lemma shows what happens when one integer is not greater or equal than another.

```
lemma (in int0) Int_ZF_1_3_L6:
```

```
  assumes A1: a∈ℤ  b∈ℤ
  shows ¬(b≤a) ⟷ a+1 ≤ b
proof
  assume A3: ¬(b≤a)
  with A1 have  a≤b by (rule Int_ZF_2_L19)
  then have a = b   ∨  a+1 ≤ b
    using Int_ZF_4_L1B by simp
  moreover from A1 A3 have a≠b by (rule Int_ZF_2_L19)
  ultimately show a+1 ≤ b by simp
next assume A4: a+1 ≤ b
  { assume b≤a
    with A4 have a+1 ≤ a by (rule Int_order_transitive)
    moreover from A1 have a ≤ a+1
      using Int_ZF_2_L12B by simp
    ultimately have a+1 = a
      by (rule Int_ZF_2_L3)
    with A1 have False using Int_ZF_1_L14 by simp
  } then show ¬(b≤a) by auto
qed
```

Another form of stating that there are no integers between integers m and $m+1$.

```
corollary (in int0) no_int_between: assumes A1: a∈ℤ  b∈ℤ
  shows b≤a ∨ a+1 ≤ b
  using A1 Int_ZF_1_3_L6 by auto
```

Another way of saying what it means that one integer is not greater or equal than another.

```
corollary (in int0) Int_ZF_1_3_L6A:
  assumes A1: a∈ℤ  b∈ℤ and A2: ¬(b≤a)
  shows a ≤ b-1
proof -
  from A1 A2 have a+1 - 1 ≤ b - 1
    using Int_ZF_1_3_L6 int_zero_one_are_int Int_ZF_1_1_L4
      int_ord_transl_inv by simp
  with A1 show a ≤ b-1
    using int_zero_one_are_int Int_ZF_1_2_L3
    by simp
qed
```

Yet another form of stating that there are nointegers between m and $m+1$.

```
lemma (in int0) no_int_between1:
  assumes A1: a≤b  and A2: a≠b
  shows
  a+1 ≤ b
  a ≤ b-1
proof -
  from A1 have T: a∈ℤ  b∈ℤ using Int_ZF_2_L1A
```

```
    by auto
  { assume b≤a
    with A1 have a=b by (rule Int_ZF_2_L3)
    with A2 have False by simp }
  then have ¬(b≤a) by auto
  with T show
    a+1 ≤ b
    a ≤ b-1
    using no_int_between Int_ZF_1_3_L6A by auto
qed
```

We can decompose proofs into three cases: $a = b$, $a \leq b - 1b$ or $a \geq b + 1b$.

```
lemma (in int0) Int_ZF_1_3_L6B: assumes A1: a∈ℤ  b∈ℤ
  shows a=b ∨ (a ≤ b-1) ∨ (b+1 ≤a)
proof -
  from A1 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L31
    by simp
  then show thesis using no_int_between1
    by auto
qed
```

A special case of `Int_ZF_1_3_L6B` when $b = 0$. This allows to split the proofs in cases $a \leq -1$, $a = 0$ and $a \geq 1$.

```
corollary (in int0) Int_ZF_1_3_L6C: assumes A1: a∈ℤ
  shows a=0 ∨ (a ≤ -1) ∨ (1≤a)
proof -
  from A1 have a=0 ∨ (a ≤ 0 -1) ∨ (0 +1 ≤a)
    using int_zero_one_are_int Int_ZF_1_3_L6B by simp
  then show thesis using Int_ZF_1_1_L4 int_zero_one_are_int
    by simp
qed
```

An integer is not less or equal zero iff it is greater or equal one.

```
lemma (in int0) Int_ZF_1_3_L7: assumes a∈ℤ
  shows ¬(a≤0) ⟷ 1 ≤ a
  using assms int_zero_one_are_int Int_ZF_1_3_L6 Int_ZF_1_1_L4
  by simp
```

Product of positive integers is positive.

```
lemma (in int0) Int_ZF_1_3_L8:
  assumes a∈ℤ  b∈ℤ
  and ¬(a≤0)  ¬(b≤0)
  shows ¬((a·b) ≤ 0)
  using assms Int_ZF_1_3_L7 Int_ZF_1_3_L3 Int_ZF_1_1_L5 Int_ZF_1_3_L7
  by simp
```

If $a \cdot b$ is nonnegative and b is positive, then a is nonnegative. Proof by contradiction.

```
lemma (in int0) Int_ZF_1_3_L9:
  assumes A1: a∈ℤ  b∈ℤ
  and A2:  ¬(b≤0) and A3: a·b ≤ 0
  shows a≤0
proof -
  { assume ¬(a≤0)
    with A1 A2 have ¬((a·b) ≤ 0) using Int_ZF_1_3_L8
      by simp
  } with A3 show a≤0 by auto
qed
```

One integer is less or equal another iff the difference is nonpositive.

```
lemma (in int0) Int_ZF_1_3_L10:
  assumes a∈ℤ  b∈ℤ
  shows a≤b ⟷ a-b ≤ 0
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L9
  by simp
```

Some conclusions from the fact that one integer is less or equal than another.

```
lemma (in int0) Int_ZF_1_3_L10A: assumes a≤b
  shows 0 ≤ b-a
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L12A
  by simp
```

We can simplify out a positive element on both sides of an inequality.

```
lemma (in int0) Int_ineq_simpl_positive:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  and A2: a·c ≤ b·c and A4: ¬(c≤0)
  shows a ≤ b
proof -
  from A1 A4 have a-b ∈  ℤ  c∈ℤ  ¬(c≤0)
    using Int_ZF_1_1_L5 by auto
  moreover from A1 A2 have (a-b)·c ≤ 0
    using Int_ZF_1_1_L5 Int_ZF_1_3_L10 Int_ZF_1_1_L6
    by simp
  ultimately have a-b ≤ 0 by (rule Int_ZF_1_3_L9)
  with A1 show a ≤ b using Int_ZF_1_3_L10 by simp
qed
```

A technical lemma about conclusion from an inequality between absolute values. This is a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L11:
  assumes A1: a∈ℤ  b∈ℤ
  and A2: ¬(abs(a) ≤ abs(b))
  shows ¬(abs(a) ≤ 0)
proof -
  { assume abs(a) ≤ 0
    moreover from A1 have 0 ≤ abs(a) using int_abs_nonneg
```

```
      by simp
    ultimately have abs(a) = 0 by (rule Int_ZF_2_L3)
    with A1 A2 have False using int_abs_nonneg by simp
  } then show  ¬(abs(a) ≤ 0) by auto
qed
```

Negative times positive is negative. This a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L12:
  assumes a≤0  and 0≤b
  shows a·b ≤ 0
  using assms Int_ZF_1_3_T1 ring1.OrdRing_ZF_1_L8
  by simp
```

We can multiply an inequality by a nonnegative number. This is a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L13:
  assumes A1: a≤b and A2: 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
  using assms Int_ZF_1_3_T1 ring1.OrdRing_ZF_1_L9 by auto
```

A technical lemma about decreasing a factor in an inequality.

```
lemma (in int0) Int_ZF_1_3_L13A:
  assumes 1≤a and b≤c and (a+1)·c ≤ d
  shows (a+1)·b ≤ d
proof -
  from assms have
    (a+1)·b ≤ (a+1)·c
    (a+1)·c ≤ d
    using Int_ZF_2_L16C Int_ZF_1_3_L13 by auto
  then show (a+1)·b ≤ d by (rule Int_order_transitive)
qed
```

We can multiply an inequality by a positive number. This is a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L13B:
  assumes A1: a≤b and A2: c∈ℤ+
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  let R = ℤ
  let A = IntegerAddition
  let M = IntegerMultiplication
  let r = IntegerOrder
  from A1 A2 have
    ring1(R, A, M, r)
```

```
    ⟨a,b⟩ ∈ r
    c ∈ PositiveSet(R, A, r)
    using Int_ZF_1_3_T1 by auto
  then show
    a·c ≤ b·c
    c·a ≤ c·b
    using ring1.OrdRing_ZF_1_L9A by auto
qed
```

A rearrangement with four integers and absolute value.

```
lemma (in int0) Int_ZF_1_3_L14:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows abs(a·b)+(abs(a)+c)·d = (d+abs(b))·abs(a)+c·d
proof -
  from A1 have T1:
    abs(a) ∈ ℤ  abs(b) ∈ ℤ
    abs(a)·abs(b) ∈ ℤ
    abs(a)·d ∈ ℤ
    c·d ∈ ℤ
    abs(b)+d ∈ ℤ
    using Int_ZF_2_L14 Int_ZF_1_1_L5 by auto
  with A1 have abs(a·b)+(abs(a)+c)·d = abs(a)·(abs(b)+d)+c·d
    using Int_ZF_1_3_L5 Int_ZF_1_1_L1 Int_ZF_1_1_L7 by simp
  with A1 T1 show thesis using Int_ZF_1_1_L5 by simp
qed
```

A technical lemma about what happens when one absolute value is not greater or equal than another.

```
lemma (in int0) Int_ZF_1_3_L15: assumes A1: m∈ℤ n∈ℤ
  and A2: ¬(abs(m) ≤ abs(n))
  shows n ≤ abs(m)  m≠0
proof -
  from A1 have T1: n ≤ abs(n)
    using Int_ZF_2_L19C by simp
  from A1 have abs(n) ∈ ℤ  abs(m) ∈ ℤ
    using Int_ZF_2_L14 by auto
  moreover note A2
  ultimately have abs(n) ≤ abs(m)
    by (rule Int_ZF_2_L19)
  with T1 show  n ≤ abs(m) by (rule Int_order_transitive)
  from A1 A2 show m≠0 using Int_ZF_2_L18 int_abs_nonneg by auto
qed
```

Negative of a nonnegative is nonpositive.

```
lemma (in int0) Int_ZF_1_3_L16: assumes A1: 0 ≤ m
  shows (-m) ≤ 0
proof -
  from A1 have (-m) ≤ (-0)
    using Int_ZF_2_L10 by simp
```

```
  then show (-m) ≤ 0 using Int_ZF_1_L11
    by simp
qed
```

Some statements about intervals centered at 0.

```
lemma (in int0) Int_ZF_1_3_L17: assumes A1: m∈ℤ
  shows
  (-abs(m)) ≤ abs(m)
  (-abs(m))..abs(m) ≠ 0
proof -
  from A1 have (-abs(m)) ≤ 0  0 ≤ abs(m)
    using int_abs_nonneg Int_ZF_1_3_L16 by auto
  then show (-abs(m)) ≤ abs(m) by (rule Int_order_transitive)
  then have abs(m) ∈ (-abs(m))..abs(m)
    using int_ord_is_refl Int_ZF_2_L1A Order_ZF_2_L2 by simp
  thus (-abs(m))..abs(m) ≠ 0 by auto
qed
```

The greater of two integers is indeed greater than both, and the smaller one is smaller that both.

```
lemma (in int0) Int_ZF_1_3_L18: assumes A1: m∈ℤ  n∈ℤ
  shows
  m ≤ GreaterOf(IntegerOrder,m,n)
  n ≤ GreaterOf(IntegerOrder,m,n)
  SmallerOf(IntegerOrder,m,n) ≤ m
  SmallerOf(IntegerOrder,m,n) ≤ n
  using assms Int_ZF_2_T1 Order_ZF_3_L2 by auto
```

If $|m| \leq n$, then $m \in -n..n$.

```
lemma (in int0) Int_ZF_1_3_L19:
  assumes A1: m∈ℤ and A2: abs(m) ≤ n
  shows
  (-n) ≤ m  m ≤ n
  m ∈ (-n)..n
  0 ≤ n
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L8
    group3.OrderedGroup_ZF_3_L8A Order_ZF_2_L1
  by auto
```

A slight generalization of the above lemma.

```
lemma (in int0) Int_ZF_1_3_L19A:
  assumes A1: m∈ℤ and A2: abs(m) ≤ n and A3: 0≤k
  shows (-(n+k)) ≤ m
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L8B
  by simp
```

Sets of integers that have absolute value bounded are bounded.

```
lemma (in int0) Int_ZF_1_3_L20:
```

```
  assumes A1: ∀x∈X. b(x) ∈ ℤ ∧ abs(b(x)) ≤ L
  shows IsBounded({b(x). x∈X},IntegerOrder)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 have
    group3(G, P, r)
    r {is total on} G
    ∀x∈X. b(x) ∈ G ∧ ⟨AbsoluteValue(G, P, r)  b(x), L⟩ ∈ r
    using Int_ZF_2_T1 by auto
  then show IsBounded({b(x). x∈X},IntegerOrder)
    by (rule group3.OrderedGroup_ZF_3_L9A)
qed
```

If a set is bounded, then the absolute values of the elements of that set are bounded.

```
lemma (in int0) Int_ZF_1_3_L20A: assumes IsBounded(A,IntegerOrder)
  shows ∃L. ∀a∈A. abs(a) ≤ L
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L10A
  by simp
```

Absolute vaues of integers from a finite image of integers are bounded by an integer.

```
lemma (in int0) Int_ZF_1_3_L20AA:
  assumes A1: {b(x). x∈ℤ} ∈ Fin(ℤ)
  shows ∃L∈ℤ. ∀x∈ℤ. abs(b(x)) ≤ L
  using assms int_not_empty Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L11A
  by simp
```

If absolute values of values of some integer function are bounded, then the image a set from the domain is a bounded set.

```
lemma (in int0) Int_ZF_1_3_L20B:
  assumes f:X→ℤ and A⊆X and ∀x∈A. abs(f(x)) ≤ L
  shows  IsBounded(f(A),IntegerOrder)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from assms have
    group3(G, P, r)
    r {is total on} G
    f:X→G
    A⊆X
    ∀x∈A. ⟨AbsoluteValue(G, P, r)(f(x)), L⟩ ∈ r
    using Int_ZF_2_T1 by auto
  then show IsBounded(f(A), r)
    by (rule group3.OrderedGroup_ZF_3_L9B)
```

qed

A special case of the previous lemma for a function from integers to integers.

```
corollary (in int0) Int_ZF_1_3_L20C:
  assumes f:ℤ→ℤ and ∀m∈ℤ. abs(f(m)) ≤ L
  shows f(ℤ) ∈ Fin(ℤ)
proof -
  from assms have f:ℤ→ℤ ℤ ⊆ ℤ  ∀m∈ℤ. abs(f(m)) ≤ L
    by auto
  then have IsBounded(f(ℤ),IntegerOrder)
    by (rule Int_ZF_1_3_L20B)
  then show f(ℤ) ∈ Fin(ℤ) using Int_bounded_iff_fin
    by simp
qed
```

A triangle inequality with three integers. Property of linearly ordered abelian groups.

```
lemma (in int0) int_triangle_ineq3:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows abs(a-b-c) ≤ abs(a) + abs(b) + abs(c)
proof -
  from A1 have T: a-b ∈ ℤ  abs(c) ∈ ℤ
    using Int_ZF_1_1_L5 Int_ZF_2_L14 by auto
  with A1 have abs(a-b-c) ≤ abs(a-b) + abs(c)
    using Int_triangle_ineq1 by simp
  moreover from A1 T have
    abs(a-b) + abs(c) ≤  abs(a) + abs(b) + abs(c)
    using Int_triangle_ineq1 int_ord_transl_inv by simp
  ultimately show thesis by (rule Int_order_transitive)
qed
```

If $a \leq c$ and $b \leq c$, then $a + b \leq 2 \cdot c$. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L21:
  assumes A1: a≤c  b≤c shows a+b ≤ 2·c
  using assms Int_ZF_1_3_T1 ring1.OrdRing_ZF_2_L6 by simp
```

If an integer a is between b and $b + c$, then $|b - a| \leq c$. Property of ordered groups.

```
lemma (in int0) Int_ZF_1_3_L22:
  assumes a≤b and c∈ℤ and b≤ c+a
  shows abs(b-a) ≤ c
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L8C
  by simp
```

An application of the triangle inequality with four integers. Property of linearly ordered abelian groups.

```
lemma (in int0) Int_ZF_1_3_L22A:
  assumes  a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
```

```
  shows abs(a-c) ≤ abs(a+b) + abs(c+d) + abs(b-d)
  using assms Int_ZF_1_T2 Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L7F
  by simp
```

If an integer a is between b and $b+c$, then $|b-a| \leq c$. Property of ordered groups. A version of Int_ZF_1_3_L22 with sligtly different assumptions.

```
lemma (in int0) Int_ZF_1_3_L23:
  assumes A1: a≤b and A2: c∈ℤ and A3: b≤ a+c
  shows abs(b-a) ≤ c
proof -
  from A1 have a ∈ ℤ
    using Int_ZF_2_L1A by simp
  with A2 A3 have b≤ c+a
    using Int_ZF_1_1_L5 by simp
  with A1 A2 show abs(b-a) ≤ c
    using Int_ZF_1_3_L22 by simp
qed
```

56.4 Maximum and minimum of a set of integers

In this section we provide some sufficient conditions for integer subsets to have extrema (maxima and minima).

Finite nonempty subsets of integers attain maxima and minima.

```
theorem (in int0) Int_fin_have_max_min:
  assumes A1: A ∈ Fin(ℤ) and A2: A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  HasAminimum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  Maximum(IntegerOrder,A) ∈ ℤ
  Minimum(IntegerOrder,A) ∈ ℤ
proof -
  from A1 have
    A=0 ∨ HasAmaximum(IntegerOrder,A) and
    A=0 ∨ HasAminimum(IntegerOrder,A)
    using Int_ZF_2_T1 Int_ZF_2_L6 Finite_ZF_1_1_T1A Finite_ZF_1_1_T1B
    by auto
  with A2 show
    HasAmaximum(IntegerOrder,A)
    HasAminimum(IntegerOrder,A)
    by auto
  from A1 A2 show
    Maximum(IntegerOrder,A) ∈ A
    Minimum(IntegerOrder,A) ∈ A
    ∀x∈A. x ≤ Maximum(IntegerOrder,A)
```

```
    ∀x∈A. Minimum(IntegerOrder,A) ≤ x
    using Int_ZF_2_T1 Finite_ZF_1_T2 by auto
  moreover from A1 have A⊆ℤ using FinD by simp
  ultimately show
    Maximum(IntegerOrder,A) ∈ ℤ
    Minimum(IntegerOrder,A) ∈ ℤ
    by auto
qed
```

Bounded nonempty integer subsets attain maximum and minimum.

```
theorem (in int0) Int_bounded_have_max_min:
  assumes IsBounded(A,IntegerOrder) and A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  HasAminimum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  Maximum(IntegerOrder,A) ∈ ℤ
  Minimum(IntegerOrder,A) ∈ ℤ
  using assms Int_fin_have_max_min Int_bounded_iff_fin
  by auto
```

Nonempty set of integers that is bounded below attains its minimum.

```
theorem (in int0) int_bounded_below_has_min:
  assumes A1: IsBoundedBelow(A,IntegerOrder) and A2: A≠0
  shows
  HasAminimum(IntegerOrder,A)
  Minimum(IntegerOrder,A) ∈ A

  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
proof -
  from A1 A2 have
    IntegerOrder {is total on} ℤ
    trans(IntegerOrder)
    IntegerOrder ⊆ ℤ×ℤ
    ∀A. IsBounded(A,IntegerOrder) ∧ A≠0 ⟶ HasAminimum(IntegerOrder,A)
    A≠0  IsBoundedBelow(A,IntegerOrder)
    using Int_ZF_2_T1 Int_ZF_2_L6 Int_ZF_2_L1B Int_bounded_have_max_min
    by auto
  then show HasAminimum(IntegerOrder,A)
    by (rule Order_ZF_4_L11)
  then show
    Minimum(IntegerOrder,A) ∈ A
    ∀x∈A. Minimum(IntegerOrder,A) ≤ x
    using Int_ZF_2_L4 Order_ZF_4_L4 by auto
qed
```

Nonempty set of integers that is bounded above attains its maximum.

```
theorem (in int0) int_bounded_above_has_max:
  assumes A1: IsBoundedAbove(A,IntegerOrder) and A2: A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Maximum(IntegerOrder,A) ∈ ℤ
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
proof -
  from A1 A2 have
    IntegerOrder {is total on} ℤ
    trans(IntegerOrder) and
    I: IntegerOrder ⊆ ℤ×ℤ and
    ∀A. IsBounded(A,IntegerOrder) ∧ A≠0 ⟶ HasAmaximum(IntegerOrder,A)
    A≠0  IsBoundedAbove(A,IntegerOrder)
    using Int_ZF_2_T1 Int_ZF_2_L6 Int_ZF_2_L1B Int_bounded_have_max_min
    by auto
  then show HasAmaximum(IntegerOrder,A)
    by (rule Order_ZF_4_L11A)
  then show
    II: Maximum(IntegerOrder,A) ∈ A and
    ∀x∈A. x ≤ Maximum(IntegerOrder,A)
    using Int_ZF_2_L4 Order_ZF_4_L3 by auto
  from I A1 have A ⊆ ℤ by (rule Order_ZF_3_L1A)
  with II show Maximum(IntegerOrder,A) ∈ ℤ by auto
qed
```

A set defined by separation over a bounded set attains its maximum and minimum.

```
lemma (in int0) Int_ZF_1_4_L1:
  assumes A1: IsBounded(A,IntegerOrder) and A2: A≠0
  and A3: ∀q∈ℤ. F(q) ∈ ℤ
  and A4: K = {F(q). q ∈ A}
  shows
  HasAmaximum(IntegerOrder,K)
  HasAminimum(IntegerOrder,K)
  Maximum(IntegerOrder,K) ∈ K
  Minimum(IntegerOrder,K) ∈ K
  Maximum(IntegerOrder,K) ∈ ℤ
  Minimum(IntegerOrder,K) ∈ ℤ
  ∀q∈A. F(q) ≤ Maximum(IntegerOrder,K)
  ∀q∈A. Minimum(IntegerOrder,K) ≤ F(q)
  IsBounded(K,IntegerOrder)
proof -
  from A1 have A ∈ Fin(ℤ) using Int_bounded_iff_fin
    by simp
  with A3 have {F(q). q ∈ A} ∈ Fin(ℤ)
    by (rule fin_image_fin)
  with A2 A4 have T1: K ∈ Fin(ℤ)  K≠0 by auto
  then show T2:
```

```
    HasAmaximum(IntegerOrder,K)
    HasAminimum(IntegerOrder,K)
    and Maximum(IntegerOrder,K) ∈ K
    Minimum(IntegerOrder,K) ∈ K
    Maximum(IntegerOrder,K) ∈ ℤ
    Minimum(IntegerOrder,K) ∈ ℤ
    using Int_fin_have_max_min by auto
  { fix q assume q∈A
    with A4 have F(q) ∈ K by auto
    with T1 have
      F(q) ≤ Maximum(IntegerOrder,K)
      Minimum(IntegerOrder,K) ≤ F(q)
      using Int_fin_have_max_min by auto
  } then show
      ∀q∈A. F(q) ≤ Maximum(IntegerOrder,K)
      ∀q∈A. Minimum(IntegerOrder,K) ≤ F(q)
    by auto
  from T2 show IsBounded(K,IntegerOrder)
    using Order_ZF_4_L7 Order_ZF_4_L8A IsBounded_def
    by simp
qed
```

A three element set has a maximume and minimum.

```
lemma (in int0) Int_ZF_1_4_L1A: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  Maximum(IntegerOrder,{a,b,c}) ∈  ℤ
  a ≤ Maximum(IntegerOrder,{a,b,c})
  b ≤ Maximum(IntegerOrder,{a,b,c})
  c ≤ Maximum(IntegerOrder,{a,b,c})
  using assms Int_ZF_2_T1 Finite_ZF_1_L2A by auto
```

Integer functions attain maxima and minima over intervals.

```
lemma (in int0) Int_ZF_1_4_L2:
  assumes A1: f:ℤ→ℤ and A2: a≤b
  shows
  maxf(f,a..b) ∈ ℤ
  ∀c ∈ a..b. f(c) ≤ maxf(f,a..b)
  ∃c ∈ a..b. f(c) = maxf(f,a..b)
  minf(f,a..b) ∈ ℤ
  ∀c ∈ a..b. minf(f,a..b) ≤ f(c)
  ∃c ∈ a..b. f(c) = minf(f,a..b)
proof -
  from A2 have T: a∈ℤ  b∈ℤ  a..b ⊆ ℤ
    using Int_ZF_2_L1A Int_ZF_2_L1B Order_ZF_2_L6
    by auto
  with A1 A2 have
    Maximum(IntegerOrder,f(a..b)) ∈ f(a..b)
    ∀x∈f(a..b). x ≤ Maximum(IntegerOrder,f(a..b))
    Maximum(IntegerOrder,f(a..b)) ∈ ℤ
```

```
    Minimum(IntegerOrder,f(a..b)) ∈ f(a..b)
    ∀x∈f(a..b). Minimum(IntegerOrder,f(a..b)) ≤ x
    Minimum(IntegerOrder,f(a..b)) ∈ ℤ
    using Int_ZF_4_L8 Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L6
      Int_fin_have_max_min by auto
  with A1 T show
    maxf(f,a..b) ∈ ℤ
    ∀c ∈ a..b. f(c) ≤ maxf(f,a..b)
    ∃c ∈ a..b. f(c) = maxf(f,a..b)
    minf(f,a..b) ∈ ℤ
    ∀c ∈ a..b. minf(f,a..b) ≤ f(c)
    ∃c ∈ a..b. f(c) = minf(f,a..b)
    using func_imagedef by auto
qed
```

56.5 The set of nonnegative integers

The set of nonnegative integers looks like the set of natural numbers. We explore that in this section. We also rephrase some lemmas about the set of positive integers known from the theory of ordered groups.

The set of positive integers is closed under addition.

```
lemma (in int0) pos_int_closed_add:
  shows ℤ₊ {is closed under} IntegerAddition
  using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L13 by simp
```

Text expended version of the fact that the set of positive integers is closed under addition

```
lemma (in int0) pos_int_closed_add_unfolded:
  assumes a∈ℤ₊  b∈ℤ₊  shows a+b ∈ ℤ₊
  using assms pos_int_closed_add IsOpClosed_def
  by simp
```

$\mathbb{Z}^+$ is bounded below.

```
lemma (in int0) Int_ZF_1_5_L1: shows
  IsBoundedBelow(ℤ⁺,IntegerOrder)
  IsBoundedBelow(ℤ₊,IntegerOrder)
  using Nonnegative_def PositiveSet_def IsBoundedBelow_def by auto
```

Subsets of $\mathbb{Z}^+$ are bounded below.

```
lemma (in int0) Int_ZF_1_5_L1A: assumes A ⊆ ℤ⁺
  shows IsBoundedBelow(A,IntegerOrder)
  using assms Int_ZF_1_5_L1 Order_ZF_3_L12 by blast
```

Subsets of $\mathbb{Z}_+$ are bounded below.

```
lemma (in int0) Int_ZF_1_5_L1B: assumes A1: A ⊆ ℤ₊
  shows IsBoundedBelow(A,IntegerOrder)
```

```
  using A1 Int_ZF_1_5_L1 Order_ZF_3_L12 by blast
```

Every nonempty subset of positive integers has a mimimum.

```
lemma (in int0) Int_ZF_1_5_L1C: assumes A ⊆ ℤ₊ and A ≠ 0
  shows
  HasAminimum(IntegerOrder,A)
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  using assms Int_ZF_1_5_L1B int_bounded_below_has_min by auto
```

Infinite subsets of Z^+ do not have a maximum - If $A \subseteq Z^+$ then for every integer we can find one in the set that is not smaller.

```
lemma (in int0) Int_ZF_1_5_L2:
  assumes A1: A ⊆ ℤ⁺  and A2: A ∉ Fin(ℤ) and A3: D∈ℤ
  shows ∃n∈A. D≤n
proof -
  { assume ∀n∈A. ¬(D≤n)
    moreover from A1 A3 have D∈ℤ  ∀n∈A. n∈ℤ
      using Nonnegative_def by auto
    ultimately have ∀n∈A. n≤D
      using Int_ZF_2_L19 by blast
    hence ∀n∈A. ⟨n,D⟩ ∈ IntegerOrder by simp
    then have IsBoundedAbove(A,IntegerOrder)
      by (rule Order_ZF_3_L10)
    with A1 have IsBounded(A,IntegerOrder)
      using Int_ZF_1_5_L1A IsBounded_def by simp
    with A2 have False using Int_bounded_iff_fin by auto
  } thus thesis by auto
qed
```

Infinite subsets of Z_+ do not have a maximum - If $A \subseteq Z_+$ then for every integer we can find one in the set that is not smaller. This is very similar to `Int_ZF_1_5_L2`, except we have $\mathbb{Z}_+$ instead of $\mathbb{Z}^+$ here.

```
lemma (in int0) Int_ZF_1_5_L2A:
  assumes A1: A ⊆ ℤ₊  and A2: A ∉ Fin(ℤ) and A3: D∈ℤ
  shows ∃n∈A. D≤n
proof -
{ assume ∀n∈A. ¬(D≤n)
    moreover from A1 A3 have D∈ℤ  ∀n∈A. n∈ℤ
      using PositiveSet_def by auto
    ultimately have ∀n∈A. n≤D
      using Int_ZF_2_L19 by blast
    hence ∀n∈A. ⟨n,D⟩ ∈ IntegerOrder by simp
    then have IsBoundedAbove(A,IntegerOrder)
      by (rule Order_ZF_3_L10)
    with A1 have IsBounded(A,IntegerOrder)
      using Int_ZF_1_5_L1B IsBounded_def by simp
    with A2 have False using Int_bounded_iff_fin by auto
```

```
  } thus thesis by auto
qed
```

An integer is either positive, zero, or its opposite is postitive.

```
lemma (in int0) Int_decomp: assumes m∈ℤ
  shows Exactly_1_of_3_holds (m=0,m∈ℤ+,(-m)∈ℤ+)
  using assms Int_ZF_2_T1 group3.OrdGroup_decomp
  by simp
```

An integer is zero, positive, or it's inverse is positive.

```
lemma (in int0) int_decomp_cases: assumes m∈ℤ
  shows m=0 ∨ m∈ℤ+ ∨ (-m) ∈ ℤ+
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L14
  by simp
```

An integer is in the positive set iff it is greater or equal one.

```
lemma (in int0) Int_ZF_1_5_L3: shows m∈ℤ+ ⟷ 1≤m
proof
  assume m∈ℤ+ then have 0≤m  m≠0
    using PositiveSet_def by auto
  then have 0+1 ≤ m
    using Int_ZF_4_L1B by auto
  then show 1≤m
    using int_zero_one_are_int Int_ZF_1_T2 group0.group0_2_L2
    by simp
next assume 1≤m
  then have m∈ℤ  0≤m  m≠0
    using Int_ZF_2_L1A Int_ZF_2_L16C by auto
  then show m∈ℤ+ using PositiveSet_def by auto
qed
```

The set of positive integers is closed under multiplication. The unfolded form.

```
lemma (in int0) pos_int_closed_mul_unfold:
  assumes a∈ℤ+  b∈ℤ+
  shows a·b ∈ ℤ+
  using assms Int_ZF_1_5_L3 Int_ZF_1_3_L3 by simp
```

The set of positive integers is closed under multiplication.

```
lemma (in int0) pos_int_closed_mul: shows
  ℤ+ {is closed under} IntegerMultiplication
  using pos_int_closed_mul_unfold IsOpClosed_def
  by simp
```

It is an overkill to prove that the ring of integers has no zero divisors this way, but why not?

```
lemma (in int0) int_has_no_zero_divs:
```

```
  shows HasNoZeroDivs(ℤ,IntegerAddition,IntegerMultiplication)
  using pos_int_closed_mul Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L3
  by simp
```

Nonnegative integers are positive ones plus zero.

```
lemma (in int0) Int_ZF_1_5_L3A: shows ℤ⁺ = ℤ₊ ∪ {0}
  using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L24 by simp
```

We can make a function smaller than any constant on a given interval of positive integers by adding another constant.

```
lemma (in int0) Int_ZF_1_5_L4:
  assumes A1: f:ℤ→ℤ and A2: K∈ℤ N∈ℤ
  shows ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
proof -
  from A2 have N≤1 ∨ 2≤N
    using int_zero_one_are_int no_int_between
    by simp
  moreover
  { assume A3: N≤1
    let C = 0
    have C ∈ ℤ using int_zero_one_are_int
      by simp
    moreover
    { fix n assume n∈ℤ₊
      then have 1 ≤ n using Int_ZF_1_5_L3
 by simp
      with A3 have N≤n by (rule Int_order_transitive)
    } then have  ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
      by auto
    ultimately have ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
      by auto }
  moreover
  { let C = K - 1 - maxf(f,1..(N-1))
    assume 2≤N
    then have 2-1 ≤ N-1
      using int_zero_one_are_int Int_ZF_1_1_L4 int_ord_transl_inv
      by simp
    then have I: 1 ≤ N-1
      using int_zero_one_are_int Int_ZF_1_2_L3 by simp
    with A1 A2 have T:
      maxf(f,1..(N-1)) ∈ ℤ  K-1 ∈ ℤ  C ∈ ℤ
      using Int_ZF_1_4_L2 Int_ZF_1_1_L5 int_zero_one_are_int
      by auto
    moreover
    { fix n assume A4: n∈ℤ₊
      { assume A5: K ≤ f(n) + C and ¬(N≤n)
 with A2 A4 have n ≤ N-1
   using PositiveSet_def Int_ZF_1_3_L6A by simp
 with A4 have n ∈ 1..(N-1)
```

```
    using Int_ZF_1_5_L3 Interval_def by auto
  with A1 I T have f(n)+C ≤ maxf(f,1..(N-1)) + C
    using Int_ZF_1_4_L2 int_ord_transl_inv by simp
  with T have f(n)+C ≤ K-1
    using Int_ZF_1_2_L3 by simp
  with A5 have K ≤  K-1
    by (rule Int_order_transitive)
  with A2 have False using Int_ZF_1_2_L3AA by simp
       } then have K ≤ f(n) + C ⟶ N≤n
  by auto
     } then have ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
       by simp
     ultimately have ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
       by auto }
  ultimately show thesis by auto
qed
```

Absolute value is identity on positive integers.

```
lemma (in int0) Int_ZF_1_5_L4A:
  assumes a∈ℤ₊ shows abs(a) = a
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L2B
  by simp
```

One and two are in $\mathbb{Z}_+$.

```
lemma (in int0) int_one_two_are_pos: shows 1 ∈ ℤ₊  2 ∈ ℤ₊
  using int_zero_one_are_int int_ord_is_refl refl_def Int_ZF_1_5_L3
  Int_ZF_2_L16B by auto
```

The image of $\mathbb{Z}_+$ by a function defined on integers is not empty.

```
lemma (in int0) Int_ZF_1_5_L5: assumes A1: f : ℤ→X
  shows f(ℤ₊) ≠ 0
proof -
  have ℤ₊ ⊆ ℤ using PositiveSet_def by auto
  with A1 show f(ℤ₊) ≠ 0
    using int_one_two_are_pos func_imagedef by auto
qed
```

If n is positive, then $n - 1$ is nonnegative.

```
lemma (in int0) Int_ZF_1_5_L6: assumes A1: n ∈ ℤ₊
  shows
  0 ≤ n-1
  0 ∈ 0..(n-1)
  0..(n-1) ⊆ ℤ
proof -
  from A1 have 1 ≤ n  (-1) ∈ ℤ
    using Int_ZF_1_5_L3 int_zero_one_are_int Int_ZF_1_1_L4
    by auto
  then have 1-1 ≤ n-1
```

```
    using int_ord_transl_inv by simp
  then show 0 ≤ n-1
    using int_zero_one_are_int Int_ZF_1_1_L4 by simp
  then show 0 ∈ 0..(n-1)
    using int_zero_one_are_int int_ord_is_refl refl_def Order_ZF_2_L1B
    by simp
  show 0..(n-1) ⊆ ℤ
    using Int_ZF_2_L1B Order_ZF_2_L6 by simp
qed
```

Intgers greater than one in $\mathbb{Z}_+$ belong to $\mathbb{Z}_+$. This is a property of ordered groups and follows from OrderedGroup_ZF_1_L19, but Isabelle's simplifier has problems using that result directly, so we reprove it specifically for integers.

```
lemma (in int0) Int_ZF_1_5_L7:  assumes a ∈ ℤ+ and a≤b
  shows b ∈ ℤ+
proof-
  from assms have 1≤a  a≤b
    using Int_ZF_1_5_L3 by auto
  then have 1≤b by (rule Int_order_transitive)
  then show b ∈ ℤ+ using Int_ZF_1_5_L3 by simp
qed
```

Adding a positive integer increases integers.

```
lemma (in int0) Int_ZF_1_5_L7A: assumes a∈ℤ  b ∈ ℤ+
  shows a ≤ a+b  a ≠ a+b  a+b ∈ ℤ
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L22
  by auto
```

For any integer m the greater of m and 1 is a positive integer that is greater or equal than m. If we add 1 to it we get a positive integer that is strictly greater than m.

```
lemma (in int0) Int_ZF_1_5_L7B: assumes a∈ℤ
  shows
  a ≤ GreaterOf(IntegerOrder,1,a)
  GreaterOf(IntegerOrder,1,a) ∈ ℤ+
  GreaterOf(IntegerOrder,1,a) + 1 ∈ ℤ+
  a ≤ GreaterOf(IntegerOrder,1,a) + 1
  a ≠ GreaterOf(IntegerOrder,1,a) + 1
  using assms int_zero_not_one Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L12
  by auto
```

The opposite of an element of $\mathbb{Z}_+$ cannot belong to $\mathbb{Z}_+$.

```
lemma (in int0) Int_ZF_1_5_L8: assumes a ∈ ℤ+
  shows (-a) ∉ ℤ+
  using assms Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L20
  by simp
```

For every integer there is one in $\mathbb{Z}_+$ that is greater or equal.

```
lemma (in int0) Int_ZF_1_5_L9: assumes a∈ℤ
  shows ∃b∈ℤ₊. a≤b
  using assms int_not_trivial Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L23
  by simp
```

A theorem about odd extensions. Recall from `OrdereGroup_ZF.thy` that the odd extension of an integer function f defined on $\mathbb{Z}_+$ is the odd function on $\mathbb{Z}$ equal to f on $\mathbb{Z}_+$. First we show that the odd extension is defined on $\mathbb{Z}$.

```
lemma (in int0) Int_ZF_1_5_L10: assumes f : ℤ₊→ℤ
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,f) : ℤ→ℤ
  using assms Int_ZF_2_T1 group3.odd_ext_props by simp
```

On $\mathbb{Z}_+$, the odd extension of f is the same as f.

```
lemma (in int0) Int_ZF_1_5_L11: assumes f : ℤ₊→ℤ and a ∈ ℤ₊ and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows g(a) = f(a)
  using assms Int_ZF_2_T1 group3.odd_ext_props by simp
```

On -$\mathbb{Z}_+$, the value of the odd extension of f is the negative of $f(-a)$.

```
lemma (in int0) Int_ZF_1_5_L12:
  assumes f : ℤ₊→ℤ and a ∈ (-ℤ₊) and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows g(a) = -(f(-a))
  using assms Int_ZF_2_T1 group3.odd_ext_props by simp
```

Odd extensions are odd on $\mathbb{Z}$.

```
lemma (in int0) int_oddext_is_odd:
  assumes f : ℤ₊→ℤ and a∈ℤ and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows g(-a) = -(g(a))
  using assms Int_ZF_2_T1 group3.oddext_is_odd by simp
```

Alternative definition of an odd function.

```
lemma (in int0) Int_ZF_1_5_L13:  assumes A1: f: ℤ→ℤ shows
  (∀a∈ℤ. f(-a) = (-f(a))) ⟷ (∀a∈ℤ. (-(f(-a))) = f(a))
  using assms Int_ZF_1_T2 group0.group0_6_L2 by simp
```

Another way of expressing the fact that odd extensions are odd.

```
lemma (in int0) int_oddext_is_odd_alt:
  assumes f : ℤ₊→ℤ and a∈ℤ and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows (-g(-a)) = g(a)
  using assms Int_ZF_2_T1 group3.oddext_is_odd_alt by simp
```

56.6 Functions with infinite limits

In this section we consider functions (integer sequences) that have infinite limits. An integer function has infinite positive limit if it is arbitrarily large for large enough arguments. Similarly, a function has infinite negative limit if it is arbitrarily small for small enough arguments. The material in this come mostly from the section in `OrderedGroup_ZF.thy` with he same title. Here we rewrite the theorems from that section in the notation we use for integers and add some results specific for the ordered group of integers.

If an image of a set by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in int0) Int_ZF_1_6_L1: assumes f: ℤ→ℤ and
  ∀a∈ℤ.∃b∈ℤ+.∀x. b≤x ⟶ a ≤ f(x) and A ⊆ ℤ and
  IsBoundedAbove(f(A),IntegerOrder)
  shows IsBoundedAbove(A,IntegerOrder)
  using assms int_not_trivial Int_ZF_2_T1 group3.OrderedGroup_ZF_7_L1
  by simp
```

If an image of a set defined by separation by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in int0) Int_ZF_1_6_L2: assumes A1: X≠0 and A2: f: ℤ→ℤ and

  A3: ∀a∈ℤ.∃b∈ℤ+.∀x. b≤x ⟶ a ≤ f(x) and
  A4: ∀x∈X. b(x) ∈ ℤ  ∧ f(b(x)) ≤ U
  shows ∃u.∀x∈X. b(x) ≤ u
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 have
    group3(G, P, r)
    r {is total on} G
    G ≠ {TheNeutralElement(G, P)}
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y. ⟨b, y⟩ ∈ r ⟶ ⟨a, f(y)⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨f(b(x)), U⟩ ∈ r
    using int_not_trivial Int_ZF_2_T1 by auto
  then have ∃u. ∀x∈X. ⟨b(x), u⟩ ∈ r by (rule group3.OrderedGroup_ZF_7_L2)
  thus thesis by simp
qed
```

If an image of a set defined by separation by a integer function with infinite negative limit is bounded below, then the set itself is bounded above. This is dual to `Int_ZF_1_6_L2`.

```
lemma (in int0) Int_ZF_1_6_L3: assumes A1: X≠0 and A2: f: ℤ→ℤ and

  A3: ∀a∈ℤ.∃b∈ℤ+.∀y. b≤y ⟶ f(-y) ≤ a and
  A4: ∀x∈X. b(x) ∈ ℤ  ∧ L ≤ f(b(x))
  shows ∃l.∀x∈X. l ≤ b(x)
proof -
  let G = ℤ
```

```
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 have
    group3(G, P, r)
    r {is total on} G
    G ≠ {TheNeutralElement(G, P)}
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y.
    ⟨b, y⟩ ∈ r ⟶ ⟨f(GroupInv(G, P)(y)),a⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨L,f(b(x))⟩ ∈ r
    using int_not_trivial Int_ZF_2_T1 by auto
  then have ∃l. ∀x∈X. ⟨l, b(x)⟩ ∈ r by (rule group3.OrderedGroup_ZF_7_L3)
  thus thesis by simp
qed
```

The next lemma combines `Int_ZF_1_6_L2` and `Int_ZF_1_6_L3` to show that if the image of a set defined by separation by a function with infinite limits is bounded, then the set itself is bounded. The proof again uses directly a fact from `OrderedGroup_ZF`.

```
lemma (in int0) Int_ZF_1_6_L4:
  assumes A1: X≠0 and A2: f: ℤ→ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x) and
  A4: ∀a∈ℤ.∃b∈ℤ₊.∀y. b≤y ⟶ f(-y) ≤ a and
  A5: ∀x∈X. b(x) ∈ ℤ ∧ f(b(x)) ≤ U ∧ L ≤ f(b(x))
  shows ∃M.∀x∈X. abs(b(x)) ≤ M
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 A5 have
    group3(G, P, r)
    r {is total on} G
    G ≠ {TheNeutralElement(G, P)}
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y. ⟨b, y⟩ ∈ r ⟶ ⟨a, f(y)⟩ ∈ r
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y.
    ⟨b, y⟩ ∈ r ⟶ ⟨f(GroupInv(G, P)(y)),a⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨L,f(b(x))⟩ ∈ r ∧ ⟨f(b(x)), U⟩ ∈ r
    using int_not_trivial Int_ZF_2_T1 by auto
  then have ∃M. ∀x∈X. ⟨AbsoluteValue(G, P, r)  b(x), M⟩ ∈ r
    by (rule group3.OrderedGroup_ZF_7_L4)
  thus thesis by simp
qed
```

If a function is larger than some constant for arguments large enough, then the image of a set that is bounded below is bounded below. This is not true for ordered groups in general, but only for those for which bounded sets are finite. This does not require the function to have infinite limit, but such functions do have this property.

```
lemma (in int0) Int_ZF_1_6_L5:
  assumes A1: f: ℤ→ℤ and A2: N∈ℤ and
  A3: ∀m. N≤m ⟶ L ≤ f(m) and
  A4: IsBoundedBelow(A,IntegerOrder)
  shows IsBoundedBelow(f(A),IntegerOrder)
proof -
  from A2 A4 have A = {x∈A. x≤N} ∪ {x∈A. N≤x}
    using Int_ZF_2_T1 Int_ZF_2_L1C Order_ZF_1_L5
    by simp
  moreover have
    f({x∈A. x≤N} ∪ {x∈A. N≤x}) =
    f{x∈A. x≤N} ∪ f{x∈A. N≤x}
    by (rule image_Un)
  ultimately have f(A) = f{x∈A. x≤N} ∪ f{x∈A. N≤x}
    by simp
  moreover have IsBoundedBelow(f{x∈A. x≤N},IntegerOrder)
  proof -
    let B = {x∈A. x≤N}
    from A4 have B ∈ Fin(ℤ)
      using Order_ZF_3_L16 Int_bounded_iff_fin by auto
    with A1 have  IsBounded(f(B),IntegerOrder)
      using Finite1_L6A Int_bounded_iff_fin by simp
    then show IsBoundedBelow(f(B),IntegerOrder)
      using IsBounded_def by simp
  qed
  moreover have IsBoundedBelow(f{x∈A. N≤x},IntegerOrder)
  proof -
    let C = {x∈A. N≤x}
    from A4 have C ⊆ ℤ using Int_ZF_2_L1C by auto
    with A1 A3 have ∀y ∈ f(C). ⟨L,y⟩ ∈ IntegerOrder
      using func_imagedef by simp
    then show IsBoundedBelow(f(C),IntegerOrder)
      by (rule Order_ZF_3_L9)
  qed
  ultimately show IsBoundedBelow(f(A),IntegerOrder)
    using Int_ZF_2_T1 Int_ZF_2_L6 Int_ZF_2_L1B Order_ZF_3_L6
    by simp
qed
```

A function that has an infinite limit can be made arbitrarily large on positive integers by adding a constant. This does not actually require the function to have infinite limit, just to be larger than a constant for arguments large enough.

```
lemma (in int0) Int_ZF_1_6_L6: assumes A1: N∈ℤ and
  A2: ∀m. N≤m ⟶ L ≤ f(m) and
  A3: f: ℤ→ℤ and A4: K∈ℤ
  shows ∃c∈ℤ. ∀n∈ℤ₊. K ≤ f(n)+c
proof -
  have IsBoundedBelow(ℤ₊,IntegerOrder)
```

```
    using Int_ZF_1_5_L1 by simp
  with A3 A1 A2 have IsBoundedBelow(f(ℤ₊),IntegerOrder)
    by (rule Int_ZF_1_6_L5)
  with A1 obtain l where I: ∀y∈f(ℤ₊). l ≤ y
    using Int_ZF_1_5_L5 IsBoundedBelow_def by auto
  let c = K-l
  from A3 have f(ℤ₊) ≠ 0 using Int_ZF_1_5_L5
    by simp
  then have ∃y. y ∈ f(ℤ₊) by (rule nonempty_has_element)
  then obtain y where y ∈ f(ℤ₊) by auto
  with A4 I have T: l ∈ ℤ  c ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  { fix n assume A5: n∈ℤ₊
    have ℤ₊ ⊆ ℤ using PositiveSet_def by auto
    with A3 I T A5 have l + c ≤ f(n) + c
      using func_imagedef int_ord_transl_inv by auto
     with I T have l + c ≤ f(n) + c
      using int_ord_transl_inv by simp
    with A4 T have K ≤  f(n) + c
      using Int_ZF_1_2_L3 by simp
  } then have ∀n∈ℤ₊. K ≤  f(n) + c by simp
  with T show thesis by auto
qed
```

If a function has infinite limit, then we can add such constant such that minimum of those arguments for which the function (plus the constant) is larger than another given constant is greater than a third constant. It is not as complicated as it sounds.

```
lemma (in int0) Int_ZF_1_6_L7:
  assumes A1: f: ℤ→ℤ and A2: K∈ℤ  N∈ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x)
  shows ∃C∈ℤ. N ≤ Minimum(IntegerOrder,{n∈ℤ₊. K ≤ f(n)+C})
proof -
  from A1 A2 have ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
    using Int_ZF_1_5_L4 by simp
  then obtain C where I: C∈ℤ and
    II: ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
    by auto
  have antisym(IntegerOrder) using Int_ZF_2_L4 by simp
  moreover have HasAminimum(IntegerOrder,{n∈ℤ₊. K ≤ f(n)+C})
  proof -
    from A2 A3 I have ∃n∈ℤ₊.∀x. n≤x ⟶ K-C ≤ f(x)
      using Int_ZF_1_1_L5 by simp
    then obtain n where
      n∈ℤ₊ and ∀x. n≤x ⟶ K-C ≤  f(x)
      by auto
    with A2 I have
      {n∈ℤ₊. K ≤ f(n)+C} ≠ 0
      {n∈ℤ₊. K ≤ f(n)+C} ⊆ ℤ₊
```

```
      using int_ord_is_refl refl_def PositiveSet_def Int_ZF_2_L9C
      by auto
    then show HasAminimum(IntegerOrder,{n∈ℤ+. K ≤ f(n)+C})
      using Int_ZF_1_5_L1C by simp
  qed
  moreover from II have
    ∀n ∈ {n∈ℤ+. K ≤ f(n)+C}. ⟨N,n⟩ ∈ IntegerOrder
    by auto
  ultimately have
    ⟨N,Minimum(IntegerOrder,{n∈ℤ+. K ≤ f(n)+C})⟩ ∈ IntegerOrder
    by (rule Order_ZF_4_L12)
  with I show thesis by auto
qed
```

For any integer m the function $k \mapsto m \cdot k$ has an infinite limit (or negative of that). This is why we put some properties of these functions here, even though they properly belong to a (yet nonexistent) section on homomorphisms. The next lemma shows that the set $\{a \cdot x : x \in Z\}$ can finite only if $a = 0$.

```
lemma (in int0) Int_ZF_1_6_L8:
  assumes A1: a∈ℤ and A2: {a·x. x∈ℤ} ∈ Fin(ℤ)
  shows a = 0
proof -
  from A1 have a=0 ∨ (a ≤ -1) ∨ (1≤a)
    using Int_ZF_1_3_L6C by simp
  moreover
  { assume a ≤ -1
    then have {a·x. x∈ℤ} ∉ Fin(ℤ)
      using int_zero_not_one Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L6
      by simp
    with A2 have False by simp }
  moreover
  { assume 1≤a
    then have {a·x. x∈ℤ} ∉ Fin(ℤ)
      using int_zero_not_one Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L5
    by simp
  with A2 have False by simp }
  ultimately show  a = 0 by auto
qed
```

56.7 Miscelaneous

In this section we put some technical lemmas needed in various other places that are hard to classify.

Suppose we have an integer expression (a meta-function)F such that $F(p)|p|$ is bounded by a linear function of $|p|$, that is for some integers A, B we have $F(p)|p| \leq A|p| + B$. We show that F is then bounded. The proof is easy, we just divide both sides by $|p|$ and take the limit (just kidding).

```
lemma (in int0) Int_ZF_1_7_L1:
  assumes A1: ∀q∈ℤ. F(q) ∈ ℤ and
  A2:  ∀q∈ℤ. F(q)·abs(q) ≤ A·abs(q) + B and
  A3: A∈ℤ  B∈ℤ
  shows ∃L. ∀p∈ℤ. F(p) ≤ L
proof -
  let I = (-abs(B))..abs(B)
  let K = {F(q). q ∈ I}
  let M = Maximum(IntegerOrder,K)
  let L = GreaterOf(IntegerOrder,M,A+1)
  from A3 A1 have C1:
    IsBounded(I,IntegerOrder)
    I ≠ 0
    ∀q∈ℤ. F(q) ∈ ℤ
    K = {F(q). q ∈ I}
    using Order_ZF_3_L11 Int_ZF_1_3_L17 by auto
  then have M ∈ ℤ by (rule Int_ZF_1_4_L1)
  with A3 have T1: M ≤ L  A+1 ≤ L
    using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_1_3_L18
    by auto
  from C1 have T2: ∀q∈I. F(q) ≤ M
    by (rule Int_ZF_1_4_L1)
  { fix p assume A4: p∈ℤ have F(p) ≤ L
    proof -
      { assume abs(p) ≤ abs(B)
 with A4 T1 T2 have F(p) ≤ M  M ≤ L
   using Int_ZF_1_3_L19 by auto
 then have F(p) ≤ L by (rule Int_order_transitive) }
      moreover
      { assume A5: ¬(abs(p) ≤ abs(B))
 from A3 A2 A4 have
   A·abs(p) ∈ ℤ  F(p)·abs(p) ≤ A·abs(p) + B
   using Int_ZF_2_L14 Int_ZF_1_1_L5 by auto
 moreover from A3 A4 A5 have B ≤ abs(p)
   using Int_ZF_1_3_L15 by simp
 ultimately have
   F(p)·abs(p) ≤ A·abs(p) + abs(p)
   using Int_ZF_2_L15A by blast
 with A3 A4 have F(p)·abs(p) ≤ (A+1)·abs(p)
   using Int_ZF_2_L14 Int_ZF_1_2_L7 by simp
 moreover from A3 A1 A4 A5 have
   F(p) ∈ ℤ  A+1 ∈ ℤ  abs(p) ∈ ℤ
   ¬(abs(p) ≤ 0)
   using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_2_L14 Int_ZF_1_3_L11
   by auto
 ultimately have F(p) ≤ A+1
   using Int_ineq_simpl_positive by simp
 moreover from T1 have  A+1 ≤ L by simp
```

```
 ultimately have F(p) ≤ L by (rule Int_order_transitive) }
      ultimately show thesis by blast
    qed
  } then have ∀p∈ℤ. F(p) ≤ L by simp
  thus thesis by auto
qed
```

A lemma about splitting (not really, there is some overlap) the $\mathbb{Z}\times\mathbb{Z}$ into six subsets (cases). The subsets are as follows: first and third quadrant, and second and fourth quadrant farther split by the $b = -a$ line.

```
lemma (in int0) int_plane_split_in6: assumes a∈ℤ  b∈ℤ
  shows
  0≤a ∧ 0≤b  ∨  a≤0 ∧ b≤0  ∨
  a≤0 ∧ 0≤b ∧ 0 ≤ a+b  ∨ a≤0 ∧ 0≤b ∧ a+b ≤ 0  ∨
  0≤a ∧ b≤0 ∧ 0 ≤ a+b  ∨  0≤a ∧ b≤0 ∧ a+b ≤ 0
  using assms Int_ZF_2_T1 group3.OrdGroup_6cases by simp

end
```

57 Division on integers

```
theory IntDiv_ZF_IML imports Int_ZF_1 ZF.IntDiv

begin
```

This theory translates some results form the Isabelle's `IntDiv.thy` theory to the notation used by IsarMathLib.

57.1 Quotient and reminder

For any integers m, n , $n > 0$ there are unique integers q, p such that $0 \leq p < n$ and $m = n \cdot q + p$. Number p in this decompsition is usually called m mod n. Standard Isabelle denotes numbers q, p as `m zdiv n` and `m zmod n`, resp., and we will use the same notation.

The next lemma is sometimes called the "quotient-reminder theorem".

```
lemma (in int0) IntDiv_ZF_1_L1: assumes m∈ℤ  n∈ℤ
  shows m = n·(m zdiv n) + (m zmod n)
  using assms Int_ZF_1_L2 raw_zmod_zdiv_equality
  by simp
```

If n is greater than 0 then `m zmod n` is between 0 and $n - 1$.

```
lemma (in int0) IntDiv_ZF_1_L2:
  assumes A1: m∈ℤ and A2: 0≤n  n≠0
  shows
  0 ≤ m zmod n
  m zmod n ≤ n    m zmod n ≠ n
```

```
  m zmod n ≤ n-1
proof -
  from A2 have T: n ∈ ℤ
    using Int_ZF_2_L1A by simp
  from A2 have #0 $< n using Int_ZF_2_L9 Int_ZF_1_L8
    by auto
  with T show
    0 ≤ m zmod n
    m zmod n ≤ n
    m zmod n ≠ n
    using pos_mod Int_ZF_1_L8 Int_ZF_1_L8A zmod_type
      Int_ZF_2_L1 Int_ZF_2_L9AA
    by auto
  then show m zmod n ≤ n-1
    using Int_ZF_4_L1B by auto
qed
```

$(m \cdot k)$ div $k = m$.

```
lemma (in int0) IntDiv_ZF_1_L3:
  assumes m∈ℤ  k∈ℤ  and k≠0
  shows
  (m·k) zdiv k = m
  (k·m) zdiv k = m
  using assms zdiv_zmult_self1 zdiv_zmult_self2
    Int_ZF_1_L8 Int_ZF_1_L2 by auto
```

The next lemma essentially translates `zdiv_mono1` from standard Isabelle to our notation.

```
lemma (in int0) IntDiv_ZF_1_L4:
  assumes A1: m ≤ k and A2: 0≤n  n≠0
  shows m zdiv n ≤  k zdiv n
proof -
  from A2 have #0 ≤ n  #0 ≠ n
    using Int_ZF_1_L8 by auto
  with A1 have
    m zdiv n $≤ k zdiv n
    m zdiv n ∈ ℤ    m zdiv k ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_2_L9 zdiv_mono1
    by auto
  then show (m zdiv n) ≤ (k zdiv n)
    using Int_ZF_2_L1 by simp
qed
```

A quotient-reminder theorem about integers greater than a given product.

```
lemma (in int0) IntDiv_ZF_1_L5:
  assumes A1: n ∈ ℤ₊ and A2: n ≤ k and A3: k·n ≤ m
  shows
  m = n·(m zdiv n) + (m zmod n)
  m = (m zdiv n)·n + (m zmod n)
```

```
  (m zmod n) ∈ 0..(n-1)
  k ≤ (m zdiv n)
  m zdiv n ∈ ℤ+
proof -
  from A2 A3 have T:
    m∈ℤ  n∈ℤ  k∈ℤ  m zdiv n ∈ ℤ
    using Int_ZF_2_L1A by auto
   then show m = n·(m zdiv n) + (m zmod n)
     using IntDiv_ZF_1_L1 by simp
   with T show m = (m zdiv n)·n + (m zmod n)
     using Int_ZF_1_L4 by simp
    from A1 have I: 0≤n  n≠0
     using PositiveSet_def by auto
   with T show (m zmod n) ∈ 0..(n-1)
    using IntDiv_ZF_1_L2 Order_ZF_2_L1
    by simp
  from A3 I have (k·n zdiv n) ≤ (m zdiv n)
    using IntDiv_ZF_1_L4 by simp
  with I T show k ≤ (m zdiv n)
    using IntDiv_ZF_1_L3 by simp
  with A1 A2 show m zdiv n ∈ ℤ+
    using Int_ZF_1_5_L7 by blast
qed


end
```

58 Integers 2

theory Int_ZF_2 **imports** func_ZF_1 Int_ZF_1 IntDiv_ZF_IML Group_ZF_3

begin

In this theory file we consider the properties of integers that are needed for the real numbers construction in Real_ZF series.

58.1 Slopes

In this section we study basic properties of slopes - the integer almost homomorphisms. The general definition of an almost homomorphism f on a group G written in additive notation requires the set $\{f(m+n) - f(m) - f(n) : m, n \in G\}$ to be finite. In this section we establish a definition that is equivalent for integers: that for all integer m, n we have $|f(m+n)-f(m)-f(n)| \leq L$ for some L.

First we extend the standard notation for integers with notation related to slopes. We define slopes as almost homomorphisms on the additive group of integers. The set of slopes is denoted $\mathcal{S}$. We also define "positive" slopes as those that take infinite number of positive values on positive integers. We write δ(s,m,n) to denote the homomorphism difference of s at m,n (i.e the expression $s(m+n)-s(m)-s(n)$). We denote maxδ(s) the maximum absolute value of homomorphism difference of s as m,n range over integers. If s is a slope, then the set of homomorphism differences is finite and this maximum exists. In Group_ZF_3 we define the equivalence relation on almost homomorphisms using the notion of a quotient group relation and use "$\approx$" to denote it. As here this symbol seems to be hogged by the standard Isabelle, we will use "$\sim$" instead "$\approx$". We show in this section that $s \sim r$ iff for some L we have $|s(m)-r(m)| \leq L$ for all integer m. The "+" denotes the first operation on almost homomorphisms. For slopes this is addition of functions defined in the natural way. The "$\circ$" symbol denotes the second operation on almost homomorphisms (see Group_ZF_3 for definition), defined for the group of integers. In short "$\circ$" is the composition of slopes. The "$^{-1}$" symbol acts as an infix operator that assigns the value $\min\{n \in Z_+ : p \leq f(n)\}$ to a pair (of sets) f and p. In application f represents a function defined on Z_+ and p is a positive integer. We choose this notation because we use it to construct the right inverse in the ring of classes of slopes and show that this ring is in fact a field. To study the homomorphism difference of the function defined by $p \mapsto f^{-1}(p)$ we introduce the symbol ε defined as $\varepsilon(f,\langle m,n\rangle) = f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$. Of course the intention is to use the fact that $\varepsilon(f,\langle m,n\rangle)$ is the homomorphism difference of the function g defined as $g(m) = f^{-1}(m)$. We also define $\gamma(s,m,n)$ as the expression $\delta(f,m,-n) + s(0) - \delta(f,n,-n)$. This is useful because of the identity $f(m-n) = \gamma(m,n) + f(m) - f(n)$ that allows to obtain bounds on the value of a slope at the difference of of two integers. For every integer m we introduce notation m^S defined by $m^E(n) = m \cdot n$. The mapping $q \mapsto q^S$ embeds integers into $\mathcal{S}$ preserving the order, (that is, maps positive integers into $\mathcal{S}_+$).

locale int1 = int0 +

fixes slopes ($\mathcal{S}$)
defines slopes_def[simp]: $\mathcal{S}$ ≡ AlmostHoms(ℤ,IntegerAddition)

fixes posslopes ($\mathcal{S}_+$)
defines posslopes_def[simp]: $\mathcal{S}_+$ ≡ {s∈$\mathcal{S}$. s(ℤ₊) ∩ ℤ₊ ∉ Fin(ℤ)}

fixes δ
defines δ_def[simp]: δ(s,m,n) ≡ s(m+n)-s(m)-s(n)

fixes maxhomdiff (maxδ)
defines maxhomdiff_def[simp]:
maxδ(s) ≡ Maximum(IntegerOrder,{abs(δ(s,m,n)). ⟨ m,n⟩ ∈ ℤ×ℤ})

fixes AlEqRel

```
  defines AlEqRel_def[simp]:
  AlEqRel ≡ QuotientGroupRel(𝒮,AlHomOp1(ℤ,IntegerAddition),FinRangeFunctions(ℤ,ℤ))

  fixes AlEq (infix ∼ 68)
  defines AlEq_def[simp]: s ∼ r ≡ ⟨ s,r⟩ ∈ AlEqRel

  fixes slope_add (infix + 70)
  defines slope_add_def[simp]: s + r ≡  AlHomOp1(ℤ,IntegerAddition)⟨ s,r⟩

  fixes slope_comp (infix ∘ 70)
  defines slope_comp_def[simp]: s ∘ r ≡  AlHomOp2(ℤ,IntegerAddition)⟨
s,r⟩

  fixes neg (-_ [90] 91)
  defines neg_def[simp]: -s ≡ GroupInv(ℤ,IntegerAddition) O s

  fixes slope_inv (infix ⁻¹ 71)
  defines slope_inv_def[simp]:
  f⁻¹(p) ≡ Minimum(IntegerOrder,{n∈ℤ₊. p ≤ f(n)})
  fixes ε
  defines ε_def[simp]:
  ε(f,p) ≡ f⁻¹(fst(p)+snd(p)) - f⁻¹(fst(p)) - f⁻¹(snd(p))

  fixes γ
  defines γ_def[simp]:
  γ(s,m,n) ≡ δ(s,m,-n) - δ(s,n,-n) + s(0)

  fixes intembed (_ˢ)
  defines intembed_def[simp]: mˢ ≡ {⟨n,m·n⟩. n∈ℤ}
```

We can use theorems proven in the group1 context.

```
lemma (in int1) Int_ZF_2_1_L1: shows group1(ℤ,IntegerAddition)
  using Int_ZF_1_T2 group1_axioms.intro group1_def by simp
```

Type information related to the homomorphism difference expression.

```
lemma (in int1) Int_ZF_2_1_L2: assumes f∈𝒮 and n∈ℤ m∈ℤ
  shows
  m+n ∈ ℤ
  f(m+n) ∈ ℤ
  f(m) ∈ ℤ   f(n) ∈ ℤ
  f(m) + f(n) ∈ ℤ
  HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩) ∈ ℤ
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_2_L4A
  by auto
```

Type information related to the homomorphism difference expression.

```
lemma (in int1) Int_ZF_2_1_L2A:
  assumes f:ℤ→ℤ and n∈ℤ  m∈ℤ
  shows
```

```
  m+n ∈ ℤ
  f(m+n) ∈ ℤ   f(m) ∈ ℤ   f(n) ∈ ℤ
  f(m) + f(n) ∈ ℤ
  HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩) ∈ ℤ
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_2_L4
  by auto
```

Slopes map integers into integers.

```
lemma (in int1) Int_ZF_2_1_L2B:
  assumes A1: f∈𝒮 and A2: m∈ℤ
  shows f(m) ∈ ℤ
proof -
  from A1 have f:ℤ→ℤ using AlmostHoms_def by simp
  with A2 show f(m) ∈ ℤ using apply_funtype by simp
qed
```

The homomorphism difference in multiplicative notation is defined as the expression $s(m \cdot n) \cdot (s(m) \cdot s(n))^{-1}$. The next lemma shows that in the additive notation used for integers the homomorphism difference is $f(m+n) - f(m) - f(n)$ which we denote as δ(f,m,n).

```
lemma (in int1) Int_ZF_2_1_L3:
  assumes f:ℤ→ℤ and m∈ℤ  n∈ℤ
  shows HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩) = δ(f,m,n)
  using assms Int_ZF_2_1_L2A Int_ZF_1_T2 group0.group0_4_L4A
    HomDiff_def by auto
```

The next formula restates the definition of the homomorphism difference to express the value an almost homomorphism on a sum.

```
lemma (in int1) Int_ZF_2_1_L3A:
  assumes A1: f∈𝒮 and A2: m∈ℤ  n∈ℤ
  shows
  f(m+n) = f(m)+(f(n)+δ(f,m,n))
proof -
  from A1 A2 have
    T: f(m)∈ ℤ  f(n) ∈ ℤ  δ(f,m,n) ∈ ℤ and
    HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩) = δ(f,m,n)
    using Int_ZF_2_1_L2 AlmostHoms_def Int_ZF_2_1_L3 by auto
  with A1 A2 show  f(m+n) = f(m)+(f(n)+δ(f,m,n))
    using Int_ZF_2_1_L3 Int_ZF_1_L3
      Int_ZF_2_1_L1 group1.Group_ZF_3_4_L1
    by simp
qed
```

The homomorphism difference of any integer function is integer.

```
lemma (in int1) Int_ZF_2_1_L3B:
  assumes f:ℤ→ℤ and m∈ℤ  n∈ℤ
  shows δ(f,m,n) ∈ ℤ
  using assms Int_ZF_2_1_L2A Int_ZF_2_1_L3 by simp
```

The value of an integer function at a sum expressed in terms of δ.

```
lemma (in int1) Int_ZF_2_1_L3C: assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows f(m+n) = δ(f,m,n) + f(n) + f(m)
proof -
  from A1 A2 have T:
    δ(f,m,n) ∈ ℤ  f(m+n) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ
    using Int_ZF_1_1_L5 apply_funtype by auto
  then show f(m+n) = δ(f,m,n) + f(n) + f(m)
    using Int_ZF_1_2_L15 by simp
qed
```

The next lemma presents two ways the set of homomorphism differences can be written.

```
lemma (in int1) Int_ZF_2_1_L4: assumes A1: f:ℤ→ℤ
  shows {abs(HomDiff(ℤ,IntegerAddition,f,x)). x ∈ ℤ×ℤ} =
  {abs(δ(f,m,n)). ⟨ m,n⟩ ∈ ℤ×ℤ}
proof -
  from A1 have ∀m∈ℤ. ∀n∈ℤ.
    abs(HomDiff(ℤ,IntegerAddition,f,⟨ m,n⟩)) = abs(δ(f,m,n))
    using Int_ZF_2_1_L3 by simp
  then show thesis by (rule ZF1_1_L4A)
qed
```

If f maps integers into integers and for all $m, n \in Z$ we have $|f(m+n) - f(m) - f(n)| \leq L$ for some L, then f is a slope.

```
lemma (in int1) Int_ZF_2_1_L5: assumes A1: f:ℤ→ℤ
  and A2: ∀m∈ℤ.∀n∈ℤ. abs(δ(f,m,n)) ≤ L
  shows f∈𝒮
proof -
  let Abs = AbsoluteValue(ℤ,IntegerAddition,IntegerOrder)
  have group3(ℤ,IntegerAddition,IntegerOrder)
    IntegerOrder {is total on} ℤ
    using Int_ZF_2_T1 by auto
  moreover from A1 A2 have
    ∀x∈ℤ×ℤ. HomDiff(ℤ,IntegerAddition,f,x) ∈ ℤ ∧
    ⟨Abs(HomDiff(ℤ,IntegerAddition,f,x)),L ⟩ ∈ IntegerOrder
    using Int_ZF_2_1_L2A Int_ZF_2_1_L3 by auto
  ultimately have
    IsBounded({HomDiff(ℤ,IntegerAddition,f,x). x∈ℤ×ℤ},IntegerOrder)
    by (rule group3.OrderedGroup_ZF_3_L9A)
  with A1 show f ∈ 𝒮 using Int_bounded_iff_fin AlmostHoms_def
    by simp
qed
```

The absolute value of homomorphism difference of a slope s does not exceed `maxδ(s)`.

```
lemma (in int1) Int_ZF_2_1_L7:
  assumes A1: s∈𝒮 and A2: n∈ℤ  m∈ℤ
```

```
  shows
  abs(δ(s,m,n)) ≤ maxδ(s)
  δ(s,m,n) ∈ ℤ   maxδ(s) ∈ ℤ
  (-maxδ(s)) ≤ δ(s,m,n)
proof -
  from A1 A2 show T: δ(s,m,n) ∈ ℤ
    using Int_ZF_2_1_L2 Int_ZF_1_1_L5 by simp
  let A = {abs(HomDiff(ℤ,IntegerAddition,s,x)). x∈ℤ×ℤ}
  let B = {abs(δ(s,m,n)). ⟨ m,n⟩ ∈ ℤ×ℤ}
  let d = abs(δ(s,m,n))
  have IsLinOrder(ℤ,IntegerOrder) using Int_ZF_2_T1
    by simp
  moreover have A ∈ Fin(ℤ)
  proof -
    have ∀k∈ℤ. abs(k) ∈ ℤ using Int_ZF_2_L14 by simp
    moreover from A1 have
      {HomDiff(ℤ,IntegerAddition,s,x). x ∈ ℤ×ℤ} ∈ Fin(ℤ)
      using AlmostHoms_def by simp
    ultimately show A ∈ Fin(ℤ) by (rule Finite1_L6C)
  qed
  moreover have A≠0 by auto
  ultimately have ∀k∈A. ⟨k,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule Finite_ZF_1_T2)
  moreover from A1 A2 have d∈A using AlmostHoms_def Int_ZF_2_1_L4
    by auto
  ultimately have d ≤ Maximum(IntegerOrder,A) by auto
  with A1 show d ≤ maxδ(s)  maxδ(s) ∈ ℤ
    using AlmostHoms_def Int_ZF_2_1_L4 Int_ZF_2_L1A
    by auto
  with T show (-maxδ(s)) ≤ δ(s,m,n)
    using Int_ZF_1_3_L19 by simp
qed
```

A useful estimate for the value of a slope at 0, plus some type information for slopes.

```
lemma (in int1) Int_ZF_2_1_L8: assumes A1: s∈𝒮
  shows
  abs(s(0)) ≤ maxδ(s)
  0 ≤ maxδ(s)
  abs(s(0)) ∈ ℤ   maxδ(s) ∈ ℤ
  abs(s(0)) + maxδ(s) ∈ ℤ
proof -
  from A1 have s(0) ∈ ℤ
    using int_zero_one_are_int Int_ZF_2_1_L2B by simp
  then have I: 0 ≤ abs(s(0))
    and abs(δ(s,0,0)) = abs(s(0))
    using int_abs_nonneg int_zero_one_are_int Int_ZF_1_1_L4
      Int_ZF_2_L17 by auto
  moreover from A1 have abs(δ(s,0,0)) ≤ maxδ(s)
```

```
    using int_zero_one_are_int Int_ZF_2_1_L7 by simp
  ultimately show II: abs(s(0)) ≤ maxδ(s)
    by simp
  with I show 0≤maxδ(s) by (rule Int_order_transitive)
  with II show
    maxδ(s) ∈ ℤ   abs(s(0)) ∈ ℤ
    abs(s(0)) + maxδ(s) ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
qed
```

Int `Group_ZF_3.thy` we show that finite range functions valued in an abelian group form a normal subgroup of almost homomorphisms. This allows to define the equivalence relation between almost homomorphisms as the relation resulting from dividing by that normal subgroup. Then we show in `Group_ZF_3_4_L12` that if the difference of f and g has finite range (actually $f(n) \cdot g(n)^{-1}$ as we use multiplicative notation in `Group_ZF_3.thy`), then f and g are equivalent. The next lemma translates that fact into the notation used in `int1` context.

```
lemma (in int1) Int_ZF_2_1_L9: assumes A1: s∈𝒮  r∈𝒮
  and A2: ∀m∈ℤ. abs(s(m)-r(m)) ≤ L
  shows s ∼ r
proof -
  from A1 A2 have
    ∀m∈ℤ. s(m)-r(m) ∈ ℤ ∧ abs(s(m)-r(m)) ≤ L
    using Int_ZF_2_1_L2B Int_ZF_1_1_L5 by simp
  then have
    IsBounded({s(n)-r(n). n∈ℤ}, IntegerOrder)
    by (rule Int_ZF_1_3_L20)
  with A1 show s ∼ r using Int_bounded_iff_fin
    Int_ZF_2_1_L1 group1.Group_ZF_3_4_L12 by simp
qed
```

A neccessary condition for two slopes to be almost equal. For slopes the definition postulates the set $\{f(m) - g(m) : m \in Z\}$ to be finite. This lemma shows that this implies that $|f(m) - g(m)|$ is bounded (by some integer) as m varies over integers. We also mention here that in this context `s ∼ r` implies that both s and r are slopes.

```
lemma (in int1) Int_ZF_2_1_L9A: assumes s ∼ r
  shows
  ∃L∈ℤ. ∀m∈ℤ. abs(s(m)-r(m)) ≤ L
  s∈𝒮  r∈𝒮
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_4_L11
    Int_ZF_1_3_L20AA QuotientGroupRel_def by auto
```

Let's recall that the relation of almost equality is an equivalence relation on the set of slopes.

```
lemma (in int1) Int_ZF_2_1_L9B: shows
```

```
  AlEqRel ⊆ 𝒮×𝒮
  equiv(𝒮,AlEqRel)
  using Int_ZF_2_1_L1 group1.Group_ZF_3_3_L3 by auto
```

Another version of sufficient condition for two slopes to be almost equal: if the difference of two slopes is a finite range function, then they are almost equal.

```
lemma (in int1) Int_ZF_2_1_L9C: assumes s∈𝒮  r∈𝒮 and
  s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
  shows
  s ∼ r
  r ∼ s
  using assms Int_ZF_2_1_L1
    group1.Group_ZF_3_2_L13 group1.Group_ZF_3_4_L12A
  by auto
```

If two slopes are almost equal, then the difference has finite range. This is the inverse of Int_ZF_2_1_L9C.

```
lemma (in int1) Int_ZF_2_1_L9D: assumes A1: s ∼ r
  shows s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  let G = ℤ
  let f = IntegerAddition
  from A1 have AlHomOp1(G, f)⟨s,GroupInv(AlmostHoms(G, f),AlHomOp1(G,
f))(r)⟩
    ∈ FinRangeFunctions(G, G)
    using Int_ZF_2_1_L1 group1.Group_ZF_3_4_L12B by auto
  with A1 show s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
    using Int_ZF_2_1_L9A Int_ZF_2_1_L1 group1.Group_ZF_3_2_L13
    by simp
qed
```

What is the value of a composition of slopes?

```
lemma (in int1) Int_ZF_2_1_L10:
  assumes s∈𝒮  r∈𝒮 and m∈ℤ
  shows (s∘r)(m) = s(r(m))  s(r(m)) ∈ ℤ
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_4_L2 by auto
```

Composition of slopes is a slope.

```
lemma (in int1) Int_ZF_2_1_L11:
  assumes s∈𝒮  r∈𝒮
  shows s∘r ∈ 𝒮
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_4_T1 by simp
```

Negative of a slope is a slope.

```
lemma (in int1) Int_ZF_2_1_L12: assumes s∈𝒮 shows -s ∈ 𝒮
  using assms Int_ZF_1_T2 Int_ZF_2_1_L1 group1.Group_ZF_3_2_L13
  by simp
```

What is the value of a negative of a slope?

```
lemma (in int1) Int_ZF_2_1_L12A:
  assumes s∈𝒮 and m∈ℤ shows (-s)(m) = -(s(m))
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_2_L5
  by simp
```

What are the values of a sum of slopes?

```
lemma (in int1) Int_ZF_2_1_L12B: assumes s∈𝒮  r∈𝒮 and m∈ℤ
  shows (s+r)(m) = s(m) + r(m)
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_2_L12
  by simp
```

Sum of slopes is a slope.

```
lemma (in int1) Int_ZF_2_1_L12C: assumes s∈𝒮  r∈𝒮
  shows s+r ∈ 𝒮
  using assms Int_ZF_2_1_L1 group1.Group_ZF_3_2_L16
  by simp
```

A simple but useful identity.

```
lemma (in int1) Int_ZF_2_1_L13:
  assumes s∈𝒮 and n∈ℤ  m∈ℤ
  shows s(n·m) + (s(m) + δ(s,n·m,m)) = s((n+1)·m)
  using assms Int_ZF_1_1_L5 Int_ZF_2_1_L2B Int_ZF_1_2_L9 Int_ZF_1_2_L7
  by simp
```

Some estimates for the absolute value of a slope at the opposite integer.

```
lemma (in int1) Int_ZF_2_1_L14: assumes A1: s∈𝒮 and A2: m∈ℤ
  shows
  s(-m) = s(0) - δ(s,m,-m) - s(m)
  abs(s(m)+s(-m)) ≤ 2·maxδ(s)
  abs(s(-m)) ≤ 2·maxδ(s) + abs(s(m))
  s(-m) ≤ abs(s(0)) + maxδ(s) - s(m)
proof -
  from A1 A2 have T:
    (-m) ∈ ℤ  abs(s(m)) ∈ ℤ  s(0) ∈ ℤ  abs(s(0)) ∈ ℤ
    δ(s,m,-m) ∈ ℤ   s(m) ∈ ℤ   s(-m) ∈ ℤ
    (-(s(m))) ∈ ℤ  s(0) - δ(s,m,-m) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_2_1_L2B Int_ZF_2_L14 Int_ZF_2_1_L2
      Int_ZF_1_1_L5 int_zero_one_are_int by auto
  with A2 show I: s(-m) = s(0) - δ(s,m,-m) - s(m)
    using Int_ZF_1_1_L4 Int_ZF_1_2_L15 by simp
  from T have abs(s(0) - δ(s,m,-m)) ≤ abs(s(0)) + abs(δ(s,m,-m))
    using Int_triangle_ineq1 by simp
  moreover from A1 A2 T have abs(s(0)) + abs(δ(s,m,-m)) ≤  2·maxδ(s)
    using Int_ZF_2_1_L7 Int_ZF_2_1_L8 Int_ZF_1_3_L21 by simp
  ultimately have abs(s(0) - δ(s,m,-m)) ≤ 2·maxδ(s)
    by (rule Int_order_transitive)
  moreover
```

```
  from I have s(m) + s(-m) = s(m) + (s(0) - δ(s,m,-m) - s(m))
    by simp
  with T have abs(s(m) + s(-m)) = abs(s(0) - δ(s,m,-m))
    using Int_ZF_1_2_L3 by simp
  ultimately show abs(s(m)+s(-m)) ≤ 2·maxδ(s)
    by simp
  from I have abs(s(-m)) = abs(s(0) - δ(s,m,-m) - s(m))
    by simp
  with T have
    abs(s(-m)) ≤ abs(s(0)) + abs(δ(s,m,-m)) + abs(s(m))
    using int_triangle_ineq3 by simp
  moreover from A1 A2 T have
    abs(s(0)) + abs(δ(s,m,-m)) + abs(s(m)) ≤ 2·maxδ(s) + abs(s(m))
    using Int_ZF_2_1_L7 Int_ZF_2_1_L8 Int_ZF_1_3_L21 int_ord_transl_inv
by simp
  ultimately show abs(s(-m)) ≤ 2·maxδ(s) + abs(s(m))
    by (rule Int_order_transitive)
  from T have s(0) - δ(s,m,-m) ≤ abs(s(0)) + abs(δ(s,m,-m))
    using Int_ZF_2_L15E by simp
  moreover from A1 A2 T have
    abs(s(0)) + abs(δ(s,m,-m)) ≤ abs(s(0)) + maxδ(s)
    using Int_ZF_2_1_L7 int_ord_transl_inv by simp
  ultimately have s(0) - δ(s,m,-m) ≤ abs(s(0)) + maxδ(s)
    by (rule Int_order_transitive)
  with T have
    s(0) - δ(s,m,-m) - s(m) ≤ abs(s(0)) + maxδ(s) - s(m)
    using int_ord_transl_inv by simp
  with I show s(-m) ≤ abs(s(0)) + maxδ(s) - s(m)
    by simp
qed
```

An identity that expresses the value of an integer function at the opposite integer in terms of the value of that function at the integer, zero, and the homomorphism difference. We have a similar identity in `Int_ZF_2_1_L14`, but over there we assume that f is a slope.

```
lemma (in int1) Int_ZF_2_1_L14A: assumes A1: f:ℤ→ℤ and A2: m∈ℤ
  shows f(-m) = (-δ(f,m,-m)) + f(0) - f(m)
proof -
  from A1 A2 have T:
    f(-m) ∈ ℤ  δ(f,m,-m) ∈ ℤ  f(0) ∈ ℤ  f(m) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_1_1_L5 int_zero_one_are_int apply_funtype

    by auto
   with A2 show f(-m) = (-δ(f,m,-m)) + f(0) - f(m)
     using Int_ZF_1_1_L4 Int_ZF_1_2_L15 by simp
qed
```

The next lemma allows to use the expression `maxf(f,0..M-1)`. Recall that `maxf(f,A)` is the maximum of (function) f on (the set) A.

```
lemma (in int1) Int_ZF_2_1_L15:
  assumes s∈𝒮 and M ∈ ℤ₊
  shows
  maxf(s,0..(M-1)) ∈ ℤ
  ∀n ∈ 0..(M-1). s(n) ≤ maxf(s,0..(M-1))
  minf(s,0..(M-1)) ∈ ℤ
  ∀n ∈ 0..(M-1). minf(s,0..(M-1)) ≤ s(n)
  using assms AlmostHoms_def Int_ZF_1_5_L6 Int_ZF_1_4_L2
  by auto
```

A lower estimate for the value of a slope at $nM + k$.

```
lemma (in int1) Int_ZF_2_1_L16:
  assumes A1: s∈𝒮  and A2: m∈ℤ and A3: M ∈ ℤ₊ and A4: k ∈ 0..(M-1)
  shows s(m·M) + (minf(s,0..(M-1))- maxδ(s)) ≤ s(m·M+k)
proof -
  from A3 have 0..(M-1) ⊆ ℤ
    using Int_ZF_1_5_L6 by simp
  with A1 A2 A3 A4 have T: m·M ∈ ℤ   k ∈ ℤ  s(m·M) ∈ ℤ
    using PositiveSet_def Int_ZF_1_1_L5  Int_ZF_2_1_L2B
    by auto
  with A1 A3 A4 have
    s(m·M) + (minf(s,0..(M-1)) - maxδ(s)) ≤ s(m·M) + (s(k) + δ(s,m·M,k))
    using Int_ZF_2_1_L15 Int_ZF_2_1_L7 int_ineq_add_sides int_ord_transl_inv
    by simp
  with A1 T show thesis using Int_ZF_2_1_L3A by simp
qed
```

Identity is a slope.

```
lemma (in int1) Int_ZF_2_1_L17: shows id(ℤ) ∈ 𝒮
  using Int_ZF_2_1_L1 group1.Group_ZF_3_4_L15 by simp
```

Simple identities about (absolute value of) homomorphism differences.

```
lemma (in int1) Int_ZF_2_1_L18:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  abs(f(n) + f(m) - f(m+n)) = abs(δ(f,m,n))
  abs(f(m) + f(n) - f(m+n)) = abs(δ(f,m,n))
  (-(f(m))) - f(n) + f(m+n) = δ(f,m,n)
  (-(f(n))) - f(m) + f(m+n) = δ(f,m,n)
  abs((-f(m+n)) + f(m) + f(n)) = abs(δ(f,m,n))
proof -
  from A1 A2 have T:
    f(m+n) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ
    f(m+n) - f(m) -  f(n)  ∈ ℤ
    (-(f(m))) ∈ ℤ
    (-f(m+n)) + f(m) + f(n) ∈ ℤ
    using apply_funtype Int_ZF_1_1_L4 Int_ZF_1_1_L5 by auto
  then have
    abs(-(f(m+n) - f(m) -  f(n))) = abs(f(m+n) - f(m) -  f(n))
```

```
    using Int_ZF_2_L17 by simp
  moreover from T have
    (-(f(m+n) - f(m) -  f(n))) = f(n) + f(m) - f(m+n)
    using Int_ZF_1_2_L9A by simp
  ultimately show abs(f(n) + f(m) - f(m+n)) = abs(δ(f,m,n))
    by simp
  moreover from T have f(n) + f(m) = f(m) + f(n)
    using Int_ZF_1_1_L5 by simp
  ultimately show abs(f(m) + f(n) - f(m+n)) = abs(δ(f,m,n))
    by simp
  from T show
    (-(f(m))) - f(n) + f(m+n) = δ(f,m,n)
    (-(f(n))) - f(m) + f(m+n) = δ(f,m,n)
    using Int_ZF_1_2_L9 by auto
  from T have
    abs((-f(m+n)) + f(m) + f(n)) =
    abs(-((-f(m+n)) + f(m) + f(n)))
    using Int_ZF_2_L17 by simp
  also from T have
    abs(-((-f(m+n)) + f(m) + f(n))) = abs(δ(f,m,n))
    using Int_ZF_1_2_L9 by simp
  finally show abs((-f(m+n)) + f(m) + f(n)) = abs(δ(f,m,n))
    by simp
qed
```

Some identities about the homomorphism difference of odd functions.

```
lemma (in int1) Int_ZF_2_1_L19:
  assumes A1: f:ℤ→ℤ and A2: ∀x∈ℤ. (-f(-x)) = f(x)
  and A3: m∈ℤ  n∈ℤ
  shows
  abs(δ(f,-m,m+n)) = abs(δ(f,m,n))
  abs(δ(f,-n,m+n)) = abs(δ(f,m,n))
  δ(f,n,-(m+n)) = δ(f,m,n)
  δ(f,m,-(m+n)) = δ(f,m,n)
  abs(δ(f,-m,-n)) = abs(δ(f,m,n))
proof -
  from A1 A2 A3 show
    abs(δ(f,-m,m+n)) = abs(δ(f,m,n))
    abs(δ(f,-n,m+n)) = abs(δ(f,m,n))
    using Int_ZF_1_2_L3 Int_ZF_2_1_L18 by auto
  from A3 have T: m+n ∈ ℤ using Int_ZF_1_1_L5 by simp
  from A1 A2 have I: ∀x∈ℤ. f(-x) = (-f(x))
    using Int_ZF_1_5_L13 by simp
  with A1 A2 A3 T show
    δ(f,n,-(m+n)) = δ(f,m,n)
    δ(f,m,-(m+n)) = δ(f,m,n)
    using Int_ZF_1_2_L3 Int_ZF_2_1_L18 by auto
  from A3 have
    abs(δ(f,-m,-n)) = abs(f(-(m+n)) - f(-m) - f(-n))
```

```
    using Int_ZF_1_1_L5 by simp
  also from A1 A2 A3 T I have ... = abs(δ(f,m,n))
    using Int_ZF_2_1_L18 by simp
  finally show abs(δ(f,-m,-n)) = abs(δ(f,m,n)) by simp
qed
```

Recall that f is a slope iff $f(m+n)-f(m)-f(n)$ is bounded as m,n ranges over integers. The next lemma is the first step in showing that we only need to check this condition as m,n ranges over positive intergers. Namely we show that if the condition holds for positive integers, then it holds if one integer is positive and the second one is nonnegative.

```
lemma (in int1) Int_ZF_2_1_L20: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L and
  A3:  m∈ℤ⁺  n∈ℤ₊
  shows
  0 ≤ L
  abs(δ(f,m,n)) ≤ L + abs(f(0))
proof -
  from A1 A2 have
    δ(f,1,1) ∈ ℤ  and abs(δ(f,1,1)) ≤ L
    using int_one_two_are_pos PositiveSet_def Int_ZF_2_1_L3B
    by auto
  then show I: 0 ≤ L using Int_ZF_1_3_L19 by simp
  from A1 A3 have T:
    n ∈ ℤ  f(n) ∈ ℤ  f(0) ∈ ℤ
    δ(f,m,n) ∈ ℤ  abs(δ(f,m,n)) ∈ ℤ
    using PositiveSet_def int_zero_one_are_int apply_funtype
      Nonnegative_def Int_ZF_2_1_L3B Int_ZF_2_L14 by auto
  from A3 have m=0 ∨ m∈ℤ₊ using Int_ZF_1_5_L3A by auto
  moreover
  { assume m = 0
    with T I have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using Int_ZF_1_1_L4 Int_ZF_1_2_L3 Int_ZF_2_L17
 int_ord_is_refl refl_def Int_ZF_2_L15F by simp }
  moreover
  { assume m∈ℤ₊
    with A2 A3 T have abs(δ(f,m,n)) ≤ L + abs(f(0))
       using int_abs_nonneg Int_ZF_2_L15F by simp }
   ultimately show abs(δ(f,m,n)) ≤ L + abs(f(0))
     by auto
qed
```

If the slope condition holds for all pairs of integers such that one integer is positive and the second one is nonnegative, then it holds when both integers are nonnegative.

```
lemma (in int1) Int_ZF_2_1_L21: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ⁺. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L and
  A3: n∈ℤ⁺  m∈ℤ⁺
```

```
  shows abs(δ(f,m,n)) ≤ L + abs(f(0))
proof -
  from A1 A2 have
    δ(f,1,1) ∈ ℤ  and abs(δ(f,1,1)) ≤ L
    using int_one_two_are_pos PositiveSet_def Nonnegative_def Int_ZF_2_1_L3B
    by auto
  then have I: 0 ≤ L using Int_ZF_1_3_L19 by simp
  from A1 A3 have T:
    m ∈ ℤ  f(m) ∈ ℤ  f(0) ∈ ℤ  (-f(0)) ∈ ℤ
    δ(f,m,n) ∈ ℤ  abs(δ(f,m,n)) ∈ ℤ
    using int_zero_one_are_int apply_funtype Nonnegative_def
      Int_ZF_2_1_L3B Int_ZF_2_L14 Int_ZF_1_1_L4 by auto
  from A3 have n=0 ∨ n∈ℤ₊ using Int_ZF_1_5_L3A by auto
  moreover
  { assume n=0
     with T have δ(f,m,n) = -f(0)
      using Int_ZF_1_1_L4 by simp
    with T have abs(δ(f,m,n)) = abs(f(0))
      using Int_ZF_2_L17 by simp
    with T have abs(δ(f,m,n)) ≤ abs(f(0))
      using int_ord_is_refl refl_def by simp
    with T I have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using Int_ZF_2_L15F by simp }
  moreover
  { assume n∈ℤ₊
    with A2 A3 T have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using int_abs_nonneg Int_ZF_2_L15F by simp }
  ultimately show  abs(δ(f,m,n)) ≤ L + abs(f(0))
    by auto
qed
```

If the homomorphism difference is bounded on $\mathbb{Z}_+\times\mathbb{Z}_+$, then it is bounded on $\mathbb{Z}^+\times\mathbb{Z}^+$.

```
lemma (in int1) Int_ZF_2_1_L22: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  shows ∃M. ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ M
proof -
  from A1 A2 have
    ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ L + abs(f(0)) + abs(f(0))
    using Int_ZF_2_1_L20 Int_ZF_2_1_L21 by simp
  then show thesis by auto
qed
```

For odd functions we can do better than in `Int_ZF_2_1_L22`: if the homomorphism difference of f is bounded on $\mathbb{Z}^+\times\mathbb{Z}^+$, then it is bounded on $\mathbb{Z}\times\mathbb{Z}$, hence f is a slope. Loong prof by splitting the $\mathbb{Z}\times\mathbb{Z}$ into six subsets.

```
lemma (in int1) Int_ZF_2_1_L23: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  and A3: ∀x∈ℤ. (-f(-x)) = f(x)
```

```
  shows  f∈𝒮
proof -
  from A1 A2 have
    ∃M.∀a∈ℤ⁺. ∀b∈ℤ⁺. abs(δ(f,a,b)) ≤ M
    by (rule Int_ZF_2_1_L22)
  then obtain M where I: ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ M
    by auto
  { fix a b assume A4: a∈ℤ  b∈ℤ
    then have
      0≤a ∧ 0≤b  ∨  a≤0 ∧ b≤0  ∨
      a≤0 ∧ 0≤b ∧ 0 ≤ a+b  ∨ a≤0 ∧ 0≤b ∧ a+b ≤ 0  ∨
      0≤a ∧ b≤0 ∧ 0 ≤ a+b  ∨  0≤a ∧ b≤0 ∧ a+b ≤ 0
      using int_plane_split_in6 by simp
    moreover
    { assume 0≤a ∧ 0≤b
      then have a∈ℤ⁺  b∈ℤ⁺
 using Int_ZF_2_L16 by auto
      with I have abs(δ(f,a,b)) ≤ M by simp }
    moreover
    { assume a≤0 ∧ b≤0
      with I have abs(δ(f,-a,-b)) ≤ M
 using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
 using Int_ZF_2_1_L19 by simp }
    moreover
    { assume a≤0 ∧ 0≤b ∧ 0 ≤ a+b
      with I have abs(δ(f,-a,a+b)) ≤ M
 using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
 using Int_ZF_2_1_L19 by simp }
    moreover
    { assume a≤0 ∧ 0≤b ∧ a+b ≤ 0
      with I have abs(δ(f,b,-(a+b))) ≤ M
 using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
 using Int_ZF_2_1_L19 by simp }
    moreover
    { assume 0≤a ∧ b≤0 ∧ 0 ≤ a+b
      with I have abs(δ(f,-b,a+b)) ≤ M
 using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
 using Int_ZF_2_1_L19 by simp }
    moreover
    { assume 0≤a ∧ b≤0 ∧ a+b ≤ 0
      with I have abs(δ(f,a,-(a+b))) ≤ M
 using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
 using Int_ZF_2_1_L19 by simp }
    ultimately have abs(δ(f,a,b)) ≤ M by auto }
```

```
  then have ∀m∈ℤ. ∀n∈ℤ. abs(δ(f,m,n)) ≤ M by simp
  with A1 show f∈𝒮 by (rule Int_ZF_2_1_L5)
qed
```

If the homomorphism difference of a function defined on positive integers is bounded, then the odd extension of this function is a slope.

```
lemma (in int1) Int_ZF_2_1_L24:
  assumes A1: f:ℤ₊→ℤ and A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,f) ∈ 𝒮
proof -
  let g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  from A1 have g : ℤ→ℤ
    using Int_ZF_1_5_L10 by simp
  moreover have ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(g,a,b)) ≤ L
  proof -
    { fix a b assume A3: a∈ℤ₊  b∈ℤ₊
      with A1 have abs(δ(f,a,b)) =  abs(δ(g,a,b))
 using pos_int_closed_add_unfolded Int_ZF_1_5_L11
 by simp
      moreover from A2 A3 have abs(δ(f,a,b)) ≤ L by simp
      ultimately have abs(δ(g,a,b)) ≤ L by simp
    } then show thesis by simp
  qed
  moreover from A1 have ∀x∈ℤ. (-g(-x)) = g(x)
    using int_oddext_is_odd_alt by simp
  ultimately show g ∈ 𝒮 by (rule Int_ZF_2_1_L23)
qed
```

Type information related to γ.

```
lemma (in int1) Int_ZF_2_1_L25:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  δ(f,m,-n) ∈ ℤ
  δ(f,n,-n) ∈ ℤ
  (-δ(f,n,-n)) ∈ ℤ
  f(0) ∈ ℤ
  γ(f,m,n)  ∈ ℤ
proof -
  from A1 A2 show T1:
    δ(f,m,-n) ∈ ℤ  f(0) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_2_1_L3B int_zero_one_are_int apply_funtype
    by auto
  from A2 have (-n) ∈ ℤ
    using Int_ZF_1_1_L4 by simp
  with A1 A2 show δ(f,n,-n) ∈ ℤ
    using Int_ZF_2_1_L3B by simp
  then show (-δ(f,n,-n)) ∈ ℤ
    using Int_ZF_1_1_L4 by simp
  with T1 show γ(f,m,n)  ∈ ℤ
```

```
    using Int_ZF_1_1_L5 by simp
qed
```

A couple of formulae involving $f(m-n)$ and $\gamma(f,m,n)$.

```
lemma (in int1) Int_ZF_2_1_L26:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  f(m-n) = γ(f,m,n) + f(m) - f(n)
  f(m-n) = γ(f,m,n) + (f(m) - f(n))
  f(m-n) + (f(n) - γ(f,m,n)) = f(m)
proof -
  from A1 A2 have T:
    (-n) ∈ ℤ  δ(f,m,-n) ∈ ℤ
    f(0) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ  (-f(n)) ∈ ℤ
    (-δ(f,n,-n)) ∈ ℤ
    (-δ(f,n,-n))  + f(0) ∈ ℤ
    γ(f,m,n) ∈ ℤ
    using  Int_ZF_1_1_L4 Int_ZF_2_1_L25 apply_funtype Int_ZF_1_1_L5
    by auto
   with A1 A2 have f(m-n) =
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0) - f(n)) + f(m)
    using Int_ZF_2_1_L3C Int_ZF_2_1_L14A by simp
  with T have f(m-n) =
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0)) + f(m) - f(n)
    using Int_ZF_1_2_L16 by simp
  moreover from T have
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0)) = γ(f,m,n)
    using Int_ZF_1_1_L7 by simp
  ultimately show  I: f(m-n) = γ(f,m,n) + f(m) - f(n)
    by simp
  then have f(m-n) + (f(n) - γ(f,m,n)) =
    (γ(f,m,n) + f(m) - f(n)) + (f(n) - γ(f,m,n))
    by simp
  moreover from T have ... = f(m) using Int_ZF_1_2_L18
    by simp
  ultimately show f(m-n) + (f(n) - γ(f,m,n)) = f(m)
    by simp
  from T have γ(f,m,n) ∈ ℤ  f(m) ∈ ℤ  (-f(n)) ∈ ℤ
    by auto
  then have
    γ(f,m,n) + f(m) + (-f(n)) =  γ(f,m,n) + (f(m) + (-f(n)))
    by (rule Int_ZF_1_1_L7)
  with I show  f(m-n) = γ(f,m,n) + (f(m) - f(n)) by simp
qed
```

A formula expressing the difference between $f(m-n-k)$ and $f(m)-f(n)-f(k)$ in terms of γ.

```
lemma (in int1) Int_ZF_2_1_L26A:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ  k∈ℤ
```

```
  shows
  f(m-n-k) - (f(m)- f(n) - f(k)) = γ(f,m-n,k) + γ(f,m,n)
proof -
  from A1 A2 have
    T: m-n ∈ ℤ γ(f,m-n,k) ∈ ℤ  f(m) - f(n) - f(k) ∈ ℤ and
    T1: γ(f,m,n) ∈ ℤ  f(m) - f(n) ∈ ℤ  (-f(k)) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_1_1_L5 Int_ZF_2_1_L25 apply_funtype
    by auto
  from A1 A2 have
    f(m-n) - f(k) = γ(f,m,n) + (f(m) - f(n)) + (-f(k))
    using Int_ZF_2_1_L26 by simp
  also from T1 have ... = γ(f,m,n) + (f(m) - f(n) + (-f(k)))
    by (rule Int_ZF_1_1_L7)
  finally have
    f(m-n) - f(k) = γ(f,m,n) + (f(m) - f(n) - f(k))
    by simp
  moreover from A1 A2 T have
    f(m-n-k) =  γ(f,m-n,k) + (f(m-n)-f(k))
    using Int_ZF_2_1_L26 by simp
  ultimately have
    f(m-n-k) - (f(m)- f(n) - f(k)) =
    γ(f,m-n,k) + ( γ(f,m,n) + (f(m) - f(n) - f(k)))
    - (f(m)- f(n) - f(k))
    by simp
  with T T1 show thesis
    using Int_ZF_1_2_L17 by simp
qed
```

If s is a slope, then $\gamma(s, m, n)$ is uniformly bounded.

```
lemma (in int1) Int_ZF_2_1_L27: assumes A1: s∈𝒮
  shows ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
proof -
  let L = maxδ(s) + maxδ(s) + abs(s(0))
  from A1 have T:
    maxδ(s) ∈ ℤ  abs(s(0)) ∈ ℤ  L ∈ ℤ
    using Int_ZF_2_1_L8 int_zero_one_are_int Int_ZF_2_1_L2B
      Int_ZF_2_L14 Int_ZF_1_1_L5 by auto
  moreover
  { fix m
    fix n
    assume A2: m∈ℤ  n∈ℤ
    with A1 have T:
      (-n) ∈ ℤ
      δ(s,m,-n) ∈ ℤ
      δ(s,n,-n) ∈ ℤ
      (-δ(s,n,-n)) ∈ ℤ
      s(0) ∈ ℤ  abs(s(0)) ∈ ℤ
      using Int_ZF_1_1_L4 AlmostHoms_def Int_ZF_2_1_L25 Int_ZF_2_L14
      by auto
```

```
    with T have
      abs(δ(s,m,-n) - δ(s,n,-n) + s(0)) ≤
      abs(δ(s,m,-n)) + abs(-δ(s,n,-n)) + abs(s(0))
      using Int_triangle_ineq3 by simp
    moreover from A1 A2 T have
      abs(δ(s,m,-n)) + abs(-δ(s,n,-n)) + abs(s(0)) ≤ L
      using Int_ZF_2_1_L7 int_ineq_add_sides int_ord_transl_inv Int_ZF_2_L17
      by simp
   ultimately have abs(δ(s,m,-n) - δ(s,n,-n) + s(0)) ≤ L
      by (rule Int_order_transitive)
    then have abs(γ(s,m,n)) ≤ L by simp }
  ultimately show ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    by auto
qed
```

If s is a slope, then $s(m) \leq s(m-1) + M$, where L does not depend on m.

```
lemma (in int1) Int_ZF_2_1_L28: assumes A1: s∈𝒮
  shows ∃M∈ℤ. ∀m∈ℤ. s(m) ≤ s(m-1) + M
proof -
  from A1 have
    ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ.abs(γ(s,m,n)) ≤ L
    using Int_ZF_2_1_L27 by simp
  then obtain L where T: L∈ℤ and ∀m∈ℤ.∀n∈ℤ.abs(γ(s,m,n)) ≤ L
    using Int_ZF_2_1_L27 by auto
  then have I: ∀m∈ℤ.abs(γ(s,m,1)) ≤ L
    using int_zero_one_are_int by simp
  let M = s(1) + L
  from A1 T have M ∈ ℤ
    using int_zero_one_are_int Int_ZF_2_1_L2B Int_ZF_1_1_L5
    by simp
  moreover
  { fix m assume A2: m∈ℤ
    with A1 have
      T1: s:ℤ→ℤ  m∈ℤ  1∈ℤ and
      T2: γ(s,m,1) ∈ ℤ  s(1) ∈ ℤ
      using int_zero_one_are_int AlmostHoms_def
 Int_ZF_2_1_L25 by auto
    from A2 T1 have T3: s(m-1) ∈ ℤ
      using Int_ZF_1_1_L5 apply_funtype by simp
    from I A2 T2 have
      (-γ(s,m,1)) ≤ abs(γ(s,m,1))
      abs(γ(s,m,1)) ≤ L
      using Int_ZF_2_L19C by auto
    then have (-γ(s,m,1)) ≤ L
      by (rule Int_order_transitive)
    with T2 T3 have
      s(m-1) + (s(1) - γ(s,m,1)) ≤ s(m-1) + M
      using int_ord_transl_inv by simp
    moreover from T1 have
```

```
      s(m-1) + (s(1) - γ(s,m,1)) = s(m)
      by (rule Int_ZF_2_1_L26)
    ultimately have s(m) ≤ s(m-1) + M  by simp  }
  ultimately show ∃M∈ℤ. ∀m∈ℤ. s(m) ≤ s(m-1) + M
    by auto
qed
```

If s is a slope, then the difference between $s(m-n-k)$ and $s(m)-s(n)-s(k)$ is uniformly bounded.

```
lemma (in int1) Int_ZF_2_1_L29: assumes A1: s∈𝒮
  shows
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
proof -
  from A1 have ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    using Int_ZF_2_1_L27 by simp
  then obtain L where I: L∈ℤ and
    II: ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    by auto
  from I have L+L ∈ ℤ
    using Int_ZF_1_1_L5 by simp
  moreover
  { fix m n k assume A2: m∈ℤ  n∈ℤ  k∈ℤ
    with A1 have T:
      m-n ∈ ℤ  γ(s,m-n,k) ∈ ℤ  γ(s,m,n) ∈ ℤ
      using Int_ZF_1_1_L5 AlmostHoms_def Int_ZF_2_1_L25
      by auto
    then have
      I: abs(γ(s,m-n,k) + γ(s,m,n)) ≤ abs(γ(s,m-n,k)) + abs(γ(s,m,n))
      using Int_triangle_ineq by simp
    from II A2 T have
      abs(γ(s,m-n,k)) ≤ L
      abs(γ(s,m,n)) ≤ L
      by auto
    then have abs(γ(s,m-n,k)) + abs(γ(s,m,n)) ≤ L+L
      using int_ineq_add_sides by simp
    with I have abs(γ(s,m-n,k) + γ(s,m,n)) ≤ L+L
      by (rule Int_order_transitive)
    moreover from A1 A2 have
      s(m-n-k) - (s(m)- s(n) - s(k)) = γ(s,m-n,k) + γ(s,m,n)
      using AlmostHoms_def Int_ZF_2_1_L26A by simp
    ultimately have
      abs(s(m-n-k) - (s(m)- s(n) - s(k))) ≤ L+L
      by simp }
  ultimately show thesis by auto
qed
```

If s is a slope, then we can find integers M,K such that $s(m-n-k) \leq s(m)-s(n)-s(k)+M$ and $s(m)-s(n)-s(k)+K \leq s(m-n-k)$, for all integer m,n,k.

```
lemma (in int1) Int_ZF_2_1_L30: assumes A1: s∈𝒮
  shows
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
  ∃K∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k)+K ≤ s(m-n-k)
proof -
  from A1 have
    ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    using Int_ZF_2_1_L29 by simp
  then obtain M where I: M∈ℤ and II:
    ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    by auto
  from I have III: (-M) ∈ ℤ using Int_ZF_1_1_L4 by simp
  { fix m n k assume A2: m∈ℤ  n∈ℤ  k∈ℤ
    with A1 have s(m-n-k) ∈ ℤ  and s(m)-s(n)-s(k) ∈ ℤ
      using Int_ZF_1_1_L5 Int_ZF_2_1_L2B by auto
    moreover from II A2 have
      abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
      by simp
    ultimately have
      s(m-n-k) ≤ s(m)-s(n)-s(k)+M ∧
      s(m)-s(n)-s(k) - M ≤ s(m-n-k)
      using Int_triangle_ineq2 by simp
  } then have
      ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
      ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k) - M ≤ s(m-n-k)
    by auto
  with I III show
    ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
    ∃K∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k)+K ≤ s(m-n-k)
    by auto
qed
```

By definition functions f, g are almost equal if $f - g^*$ is bounded. In the next lemma we show it is sufficient to check the boundedness on positive integers.

```
lemma (in int1) Int_ZF_2_1_L31: assumes A1: s∈𝒮  r∈𝒮
  and A2: ∀m∈ℤ₊. abs(s(m)-r(m)) ≤ L
  shows s ∼ r
proof -
  let a = abs(s(0) - r(0))
  let c = 2·maxδ(s) + 2·maxδ(r) + L
  let M = Maximum(IntegerOrder,{a,L,c})
  from A2 have abs(s(1)-r(1)) ≤ L
    using int_one_two_are_pos by simp
  then have T: L∈ℤ using Int_ZF_2_L1A by simp
  moreover from A1 have a ∈ ℤ
    using int_zero_one_are_int Int_ZF_2_1_L2B
      Int_ZF_1_1_L5 Int_ZF_2_L14 by simp
  moreover from A1 T have c ∈ ℤ
```

```
    using Int_ZF_2_1_L8 int_two_three_are_int Int_ZF_1_1_L5
    by simp
  ultimately have
    I: a ≤ M and
    II: L ≤ M and
    III: c ≤ M
    using Int_ZF_1_4_L1A by auto

  { fix m assume A5: m∈ℤ
    with A1 have T:
      s(m) ∈ ℤ  r(m) ∈ ℤ  s(m) - r(m) ∈ ℤ
      s(-m) ∈ ℤ  r(-m) ∈ ℤ
      using Int_ZF_2_1_L2B Int_ZF_1_1_L4 Int_ZF_1_1_L5
      by auto
    from A5 have m=0 ∨ m∈ℤ₊ ∨ (-m) ∈ ℤ₊
      using int_decomp_cases by simp
    moreover
    { assume m=0
      with I have abs(s(m) - r(m)) ≤ M
by simp }
    moreover
    { assume m∈ℤ₊
      with A2 II have
abs(s(m)-r(m)) ≤ L and L≤M
by auto
      then have abs(s(m)-r(m)) ≤ M
by (rule Int_order_transitive) }
    moreover
    { assume A6: (-m) ∈ ℤ₊
      from T have abs(s(m)-r(m)) ≤
abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m))
using Int_ZF_1_3_L22A by simp
      moreover
      from A1 A2 III A5 A6 have
abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m)) ≤ c
c ≤ M
using Int_ZF_2_1_L14 int_ineq_add_sides by auto
      then have
abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m)) ≤ M
by (rule Int_order_transitive)
      ultimately have  abs(s(m)-r(m)) ≤ M
by (rule Int_order_transitive) }
    ultimately have abs(s(m) - r(m)) ≤ M
      by auto
  } then have ∀m∈ℤ. abs(s(m)-r(m)) ≤ M
    by simp
  with A1 show s ∼ r by (rule Int_ZF_2_1_L9)
qed
```

A sufficient condition for an odd slope to be almost equal to identity: If for all positive integers the value of the slope at m is between m and m plus some constant independent of m, then the slope is almost identity.

```
lemma (in int1) Int_ZF_2_1_L32: assumes A1: s∈𝒮  M∈ℤ
  and A2: ∀m∈ℤ₊. m ≤ s(m) ∧ s(m) ≤ m+M
  shows s ∼ id(ℤ)
proof -
  let r = id(ℤ)
  from A1 have s∈𝒮  r ∈ 𝒮
    using Int_ZF_2_1_L17 by auto
  moreover from A1 A2 have ∀m∈ℤ₊. abs(s(m)-r(m)) ≤ M
    using Int_ZF_1_3_L23 PositiveSet_def id_conv by simp
  ultimately show s ∼ id(ℤ) by (rule Int_ZF_2_1_L31)
qed
```

A lemma about adding a constant to slopes. This is actually proven in `Group_ZF_3_5_L1`, in `Group_ZF_3.thy` here we just refer to that lemma to show it in notation used for integers. Unfortunately we have to use raw set notation in the proof.

```
lemma (in int1) Int_ZF_2_1_L33:
  assumes A1: s∈𝒮 and A2: c∈ℤ and
  A3: r = {⟨m,s(m)+c⟩. m∈ℤ}
  shows
  ∀m∈ℤ. r(m) = s(m)+c
  r∈𝒮
  s ∼ r
proof -
  let G = ℤ
  let f = IntegerAddition
  let AH = AlmostHoms(G, f)
  from assms have I:
    group1(G, f)
    s ∈ AlmostHoms(G, f)
    c ∈ G
    r = {⟨x, f⟨s(x), c⟩⟩. x ∈ G}
    using Int_ZF_2_1_L1 by auto
  then have ∀x∈G. r(x) = f⟨s(x),c⟩
    by (rule group1.Group_ZF_3_5_L1)
  moreover from I have r ∈ AlmostHoms(G, f)
    by (rule group1.Group_ZF_3_5_L1)
  moreover from I have
    ⟨s, r⟩ ∈ QuotientGroupRel(AlmostHoms(G, f), AlHomOp1(G, f), FinRangeFunctions(G,
G))
    by (rule group1.Group_ZF_3_5_L1)
  ultimately show
    ∀m∈ℤ. r(m) = s(m)+c
    r∈𝒮
    s ∼ r
    by auto
```

qed

58.2 Composing slopes

Composition of slopes is not commutative. However, as we show in this section if f and g are slopes then the range of $f \circ g - g \circ f$ is bounded. This allows to show that the multiplication of real numbers is commutative.

Two useful estimates.

```
lemma (in int1) Int_ZF_2_2_L1:
  assumes A1: f:ℤ→ℤ and A2: p∈ℤ  q∈ℤ
  shows
  abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
  abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
proof -
  let R = ℤ
  let A = IntegerAddition
  let M = IntegerMultiplication
  let I = GroupInv(R, A)
  let a = f((p+1)·q)
  let b = p
  let c = f(q)
  let d = f(p·q)
  from A1 A2 have T1:
    ring0(R, A, M)  a ∈ R  b ∈ R  c ∈ R  d ∈ R
    using  Int_ZF_1_1_L2 int_zero_one_are_int Int_ZF_1_1_L5 apply_funtype

    by auto
  then have
    A⟨a,I(M⟨A⟨b, TheNeutralElement(R, M)⟩,c⟩)⟩ =
    A⟨A⟨A⟨a,I(d)⟩,I(c)⟩,A⟨d, I(M⟨b, c⟩)⟩⟩
    by (rule ring0.Ring_ZF_2_L2)
  with A2 have
    f((p+1)·q)-(p+1)·f(q) = δ(f,p·q,q)+(f(p·q)-p·f(q))
    using int_zero_one_are_int Int_ZF_1_1_L1 Int_ZF_1_1_L4 by simp
  moreover from A1 A2 T1 have δ(f,p·q,q) ∈ ℤ f(p·q)-p·f(q) ∈ ℤ
    using Int_ZF_1_1_L5 apply_funtype by auto
  ultimately show
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
    using Int_triangle_ineq by simp
  from A1 A2 have T1:
    f((p-1)·q) ∈ ℤ   p∈ℤ  f(q) ∈ ℤ   f(p·q) ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 apply_funtype by auto
  then have
    f((p-1)·q)-(p-1)·f(q) = (f(p·q)-p·f(q))-(f(p·q)-f((p-1)·q)-f(q))
    by (rule Int_ZF_1_2_L6)
  with A2 have f((p-1)·q)-(p-1)·f(q) = (f(p·q)-p·f(q))-δ(f,(p-1)·q,q)
    using Int_ZF_1_2_L7 by simp
  moreover from A1 A2 have
```

```
    f(p·q)-p·f(q) ∈ ℤ   δ(f,(p-1)·q,q) ∈ ℤ
    using Int_ZF_1_1_L5 int_zero_one_are_int apply_funtype by auto
  ultimately show
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
    using Int_triangle_ineq1 by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p|+1)\cdot$`maxδ(f)`. The proof is by induction on p and the next lemma is the induction step for the case when $0 \leq p$.

```
lemma (in int1) Int_ZF_2_2_L2:
  assumes A1: f∈𝒮 and A2: 0≤p  q∈ℤ
  and A3: abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
  shows
  abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
proof -
  from A2 have q∈ℤ  p·q ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  with A1 have I: abs(δ(f,p·q,q)) ≤ maxδ(f) by (rule Int_ZF_2_1_L7)
  moreover note A3
  moreover from A1 A2 have
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
    using AlmostHoms_def Int_ZF_2_L1A Int_ZF_2_2_L1 by simp
  ultimately have
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ maxδ(f)+(abs(p)+1)·maxδ(f)
    by (rule Int_ZF_2_L15)
  moreover from I A2 have
    maxδ(f)+(abs(p)+1)·maxδ(f) = (abs(p+1)+ 1)·maxδ(f)
    using Int_ZF_2_L1A Int_ZF_1_2_L2 by simp
  ultimately show
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
    by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p|+1)\cdot$`maxδ`. The proof is by induction on p and the next lemma is the induction step for the case when $p \leq 0$.

```
lemma (in int1) Int_ZF_2_2_L3:
  assumes A1: f∈𝒮 and A2: p≤0  q∈ℤ
  and A3: abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
  shows  abs(f((p-1)·q)-(p-1)·f(q)) ≤ (abs(p-1)+ 1)·maxδ(f)
proof -
  from A2 have q∈ℤ  (p-1)·q ∈ ℤ
    using Int_ZF_2_L1A int_zero_one_are_int Int_ZF_1_1_L5 by auto
  with A1 have I: abs(δ(f,(p-1)·q,q)) ≤ maxδ(f) by (rule Int_ZF_2_1_L7)
  moreover note A3
  moreover from A1 A2 have
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
    using AlmostHoms_def Int_ZF_2_L1A Int_ZF_2_2_L1 by simp
```

```
  ultimately have
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ maxδ(f)+(abs(p)+1)·maxδ(f)
    by (rule Int_ZF_2_L15)
  with I A2 show thesis using Int_ZF_2_L1A Int_ZF_1_2_L5 by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1) \cdot \mathtt{max}\delta(f)$. Proof by cases on $0 \leq p$.

```
lemma (in int1) Int_ZF_2_2_L4:
  assumes A1: f∈𝒮 and A2: p∈ℤ q∈ℤ
  shows abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
proof -
  { assume 0≤p
    moreover from A1 A2 have abs(f(0·q)-0·f(q)) ≤ (abs(0)+1)·maxδ(f)
      using int_zero_one_are_int Int_ZF_2_1_L2B Int_ZF_1_1_L4
 Int_ZF_2_1_L8 Int_ZF_2_L18 by simp
    moreover from A1 A2 have
      ∀p. 0≤p ∧ abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) ⟶
      abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
      using Int_ZF_2_2_L2 by simp
    ultimately have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
      by (rule Induction_on_int) }
  moreover
  { assume ¬(0≤p)
    with A2 have p≤0 using Int_ZF_2_L19A by simp
    moreover from A1 A2 have abs(f(0·q)-0·f(q)) ≤ (abs(0)+1)·maxδ(f)
      using int_zero_one_are_int Int_ZF_2_1_L2B Int_ZF_1_1_L4
 Int_ZF_2_1_L8 Int_ZF_2_L18 by simp
    moreover from A1 A2 have
      ∀p. p≤0 ∧ abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) ⟶
      abs(f((p-1)·q)-(p-1)·f(q)) ≤ (abs(p-1)+ 1)·maxδ(f)
      using Int_ZF_2_2_L3 by simp
    ultimately have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
      by (rule Back_induct_on_int) }
  ultimately show thesis by blast
qed
```

The next elegant result is Lemma 7 in the Arthan's paper [2].

```
lemma (in int1) Arthan_Lem_7:
 assumes A1: f∈𝒮 and A2: p∈ℤ  q∈ℤ
  shows abs(q·f(p)-p·f(q)) ≤ (abs(p)+abs(q)+2)·maxδ(f)
proof -
  from A1 A2 have T:
    q·f(p)-f(p·q) ∈ ℤ
    f(p·q)-p·f(q) ∈ ℤ
    f(q·p) ∈ ℤ  f(p·q) ∈ ℤ
    q·f(p) ∈ ℤ  p·f(q) ∈ ℤ
    maxδ(f) ∈ ℤ
    abs(q) ∈ ℤ  abs(p) ∈ ℤ
```

```
    using Int_ZF_1_1_L5 Int_ZF_2_1_L2B Int_ZF_2_1_L7 Int_ZF_2_L14 by auto
  moreover have abs(q·f(p)-f(p·q)) ≤ (abs(q)+1)·maxδ(f)
  proof -
    from A1 A2 have abs(f(q·p)-q·f(p)) ≤ (abs(q)+1)·maxδ(f)
      using Int_ZF_2_2_L4 by simp
    with T A2 show thesis
      using Int_ZF_2_L20 Int_ZF_1_1_L5 by simp
  qed
  moreover from A1 A2 have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
    using Int_ZF_2_2_L4 by simp
  ultimately have
    abs(q·f(p)-f(p·q)+(f(p·q)-p·f(q))) ≤ (abs(q)+1)·maxδ(f)+(abs(p)+1)·maxδ(f)
    using Int_ZF_2_L21 by simp
  with T show thesis using Int_ZF_1_2_L9 int_zero_one_are_int Int_ZF_1_2_L10
    by simp
qed
```

This is Lemma 8 in the Arthan's paper.

```
lemma (in int1) Arthan_Lem_8: assumes A1: f∈𝒮
  shows ∃A B. A∈ℤ ∧ B∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B)
proof -
  let A = maxδ(f) + abs(f(1))
  let B = 3·maxδ(f)
  from A1 have A∈ℤ B∈ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_2_1_L2B
      Int_ZF_2_1_L7 Int_ZF_2_L14 by auto
  moreover have ∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B
  proof
    fix p assume A2: p∈ℤ
    with A1 have T:
      f(p) ∈ ℤ  abs(p) ∈ ℤ  f(1) ∈ ℤ
      p·f(1) ∈ ℤ  3∈ℤ  maxδ(f) ∈ ℤ
      using Int_ZF_2_1_L2B Int_ZF_2_L14 int_zero_one_are_int
 Int_ZF_1_1_L5 Int_ZF_2_1_L7 by auto
    from A1 A2 have
      abs(1·f(p)-p·f(1)) ≤ (abs(p)+abs(1)+2)·maxδ(f)
      using int_zero_one_are_int Arthan_Lem_7 by simp
    with T have abs(f(p)) ≤ abs(p·f(1))+(abs(p)+3)·maxδ(f)
      using Int_ZF_2_L16A Int_ZF_1_1_L4 Int_ZF_1_2_L11
 Int_triangle_ineq2 by simp
    with A2 T show abs(f(p)) ≤ A·abs(p)+B
      using Int_ZF_1_3_L14 by simp
  qed
  ultimately show thesis by auto
qed
```

If f and g are slopes, then $f \circ g$ is equivalent (almost equal) to $g \circ f$. This is Theorem 9 in Arthan's paper [2].

```
theorem (in int1) Arthan_Th_9: assumes A1: f∈𝒮  g∈𝒮
```

```
  shows f∘g ∼ g∘f
proof -
  from A1 have
    ∃A B. A∈ℤ ∧ B∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B)
    ∃C D. C∈ℤ ∧ D∈ℤ ∧ (∀p∈ℤ. abs(g(p)) ≤ C·abs(p)+D)
    using Arthan_Lem_8 by auto
  then obtain A B C D where D1: A∈ℤ B∈ℤ C∈ℤ D∈ℤ and D2:
    ∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B
    ∀p∈ℤ. abs(g(p)) ≤ C·abs(p)+D
    by auto
  let E = maxδ(g)·(A+1) + maxδ(f)·(C+1)
  let F = (B·maxδ(g) + 2·maxδ(g)) + (D·maxδ(f) + 2·maxδ(f))
  { fix p assume A2: p∈ℤ
    with A1 have T1:
      g(p) ∈ ℤ  f(p) ∈ ℤ  abs(p) ∈ ℤ  2 ∈ ℤ
      f(g(p)) ∈ ℤ  g(f(p)) ∈ ℤ  f(g(p)) - g(f(p)) ∈ ℤ
      p·f(g(p)) ∈ ℤ  p·g(f(p)) ∈ ℤ
      abs(f(g(p))-g(f(p))) ∈ ℤ
      using Int_ZF_2_1_L2B Int_ZF_2_1_L10 Int_ZF_1_1_L5 Int_ZF_2_L14 int_two_three_are_int
      by auto
    with A1 A2 have
      abs((f(g(p))-g(f(p)))·p) ≤
      (abs(p)+abs(f(p))+2)·maxδ(g) + (abs(p)+abs(g(p))+2)·maxδ(f)
      using Arthan_Lem_7 Int_ZF_1_2_L10A Int_ZF_1_2_L12 by simp
    moreover have
      (abs(p)+abs(f(p))+2)·maxδ(g) + (abs(p)+abs(g(p))+2)·maxδ(f) ≤
      ((maxδ(g)·(A+1) + maxδ(f)·(C+1)))·abs(p) +
      ((B·maxδ(g) + 2·maxδ(g)) + (D·maxδ(f) + 2·maxδ(f)))
    proof -
      from D2 A2 T1 have
abs(p)+abs(f(p))+2 ≤ abs(p)+(A·abs(p)+B)+2
abs(p)+abs(g(p))+2 ≤ abs(p)+(C·abs(p)+D)+2
using Int_ZF_2_L15C by auto
      with A1 have
(abs(p)+abs(f(p))+2)·maxδ(g) ≤ (abs(p)+(A·abs(p)+B)+2)·maxδ(g)
(abs(p)+abs(g(p))+2)·maxδ(f) ≤ (abs(p)+(C·abs(p)+D)+2)·maxδ(f)
using Int_ZF_2_1_L8 Int_ZF_1_3_L13 by auto
      moreover from A1 D1 T1 have
(abs(p)+(A·abs(p)+B)+2)·maxδ(g) =
maxδ(g)·(A+1)·abs(p) + (B·maxδ(g) + 2·maxδ(g))
(abs(p)+(C·abs(p)+D)+2)·maxδ(f) =
maxδ(f)·(C+1)·abs(p) + (D·maxδ(f) + 2·maxδ(f))
using Int_ZF_2_1_L8 Int_ZF_1_2_L13 by auto
      ultimately have
(abs(p)+abs(f(p))+2)·maxδ(g) + (abs(p)+abs(g(p))+2)·maxδ(f) ≤
(maxδ(g)·(A+1)·abs(p) + (B·maxδ(g) + 2·maxδ(g))) +
(maxδ(f)·(C+1)·abs(p) + (D·maxδ(f) + 2·maxδ(f)))
using int_ineq_add_sides by simp
      moreover from A1 A2 D1 have abs(p) ∈ ℤ
```

```
 maxδ(g)·(A+1) ∈ ℤ  B·maxδ(g) + 2·maxδ(g) ∈ ℤ
 maxδ(f)·(C+1) ∈ ℤ  D·maxδ(f) + 2·maxδ(f) ∈ ℤ
 using Int_ZF_2_L14 Int_ZF_2_1_L8 int_zero_one_are_int
   Int_ZF_1_1_L5 int_two_three_are_int by auto
      ultimately show thesis using Int_ZF_1_2_L14 by simp
    qed
    ultimately have
      abs((f(g(p))-g(f(p)))·p) ≤ E·abs(p) + F
      by (rule Int_order_transitive)
    with A2 T1 have
      abs(f(g(p))-g(f(p)))·abs(p) ≤ E·abs(p) + F
      abs(f(g(p))-g(f(p))) ∈ ℤ
      using Int_ZF_1_3_L5 by auto
  } then have
      ∀p∈ℤ. abs(f(g(p))-g(f(p))) ∈ ℤ
      ∀p∈ℤ. abs(f(g(p))-g(f(p)))·abs(p) ≤ E·abs(p) + F
    by auto
  moreover from A1 D1 have E ∈ ℤ  F ∈ ℤ
    using int_zero_one_are_int int_two_three_are_int Int_ZF_2_1_L8 Int_ZF_1_1_L5
    by auto
  ultimately have
    ∃L. ∀p∈ℤ. abs(f(g(p))-g(f(p))) ≤ L
    by (rule Int_ZF_1_7_L1)
  with A1 obtain L where ∀p∈ℤ. abs((f∘g)(p)-(g∘f)(p)) ≤ L
    using Int_ZF_2_1_L10 by auto
  moreover from A1 have f∘g ∈ 𝒮  g∘f ∈ 𝒮
    using Int_ZF_2_1_L11 by auto
  ultimately show f∘g ∼ g∘f using Int_ZF_2_1_L9 by auto
qed

end
```

59 Integers 3

theory Int_ZF_3 **imports** Int_ZF_2

begin

This theory is a continuation of Int_ZF_2. We consider here the properties of slopes (almost homomorphisms on integers) that allow to define the order relation and multiplicative inverse on real numbers. We also prove theorems that allow to show completeness of the order relation of real numbers we define in Real_ZF.

59.1 Positive slopes

This section provides background material for defining the order relation on real numbers.

Positive slopes are functions (of course.)

```
lemma (in int1) Int_ZF_2_3_L1: assumes A1: f∈𝒮₊ shows f:ℤ→ℤ
  using assms AlmostHoms_def PositiveSet_def by simp
```

A small technical lemma to simplify the proof of the next theorem.

```
lemma (in int1) Int_ZF_2_3_L1A:
  assumes A1: f∈𝒮₊ and A2: ∃n ∈ f(ℤ₊) ∩ ℤ₊. a≤n
  shows ∃M∈ℤ₊. a ≤ f(M)
proof -
 from A1 have f:ℤ→ℤ  ℤ₊ ⊆ ℤ
    using AlmostHoms_def PositiveSet_def by auto
 with A2 show thesis using func_imagedef by auto
qed
```

The next lemma is Lemma 3 in the Arthan's paper.

```
lemma (in int1) Arthan_Lem_3:
  assumes A1: f∈𝒮₊ and A2: D ∈ ℤ₊
  shows ∃M∈ℤ₊. ∀m∈ℤ₊. (m+1)·D ≤ f(m·M)
proof -
  let E = maxδ(f) + D
  let A = f(ℤ₊) ∩ ℤ₊
  from A1 A2 have I: D≤E
    using Int_ZF_1_5_L3 Int_ZF_2_1_L8 Int_ZF_2_L1A Int_ZF_2_L15D
    by simp
  from A1 A2 have A ⊆ ℤ₊  A ∉ Fin(ℤ)  2·E ∈ ℤ
    using int_two_three_are_int Int_ZF_2_1_L8 PositiveSet_def Int_ZF_1_1_L5
    by auto
  with A1 have ∃M∈ℤ₊.  2·E ≤ f(M)
    using Int_ZF_1_5_L2A Int_ZF_2_3_L1A by simp
  then obtain M where II: M∈ℤ₊  and III: 2·E ≤ f(M)
    by auto
  { fix m assume m∈ℤ₊ then have A4: 1≤m
      using Int_ZF_1_5_L3 by simp
    moreover from II III have (1+1) ·E ≤ f(1·M)
      using PositiveSet_def Int_ZF_1_1_L4 by simp
    moreover have ∀k.
      1≤k ∧ (k+1)·E ≤ f(k·M) ⟶ (k+1+1)·E ≤ f((k+1)·M)
    proof -
      { fix k assume A5: 1≤k  and A6: (k+1)·E ≤ f(k·M)
 with A1 A2 II have T:
   k∈ℤ  M∈ℤ  k+1 ∈ ℤ  E∈ℤ  (k+1)·E ∈ ℤ  2·E ∈ ℤ
   using Int_ZF_2_L1A PositiveSet_def int_zero_one_are_int
     Int_ZF_1_1_L5 Int_ZF_2_1_L8 by auto
 from A1 A2 A5 II have
   δ(f,k·M,M) ∈ ℤ  abs(δ(f,k·M,M)) ≤ maxδ(f)  0≤D
   using Int_ZF_2_L1A PositiveSet_def Int_ZF_1_1_L5
     Int_ZF_2_1_L7 Int_ZF_2_L16C by auto
 with III A6 have
   (k+1)·E + (2·E - E) ≤ f(k·M) + (f(M) + δ(f,k·M,M))
```

```
    using Int_ZF_1_3_L19A int_ineq_add_sides by simp
  with A1 T have (k+1+1)·E ≤ f((k+1)·M)
    using Int_ZF_1_1_L1 int_zero_one_are_int Int_ZF_1_1_L4
      Int_ZF_1_2_L11 Int_ZF_2_1_L13 by simp
       } then show thesis by simp
     qed
     ultimately have (m+1)·E ≤ f(m·M) by (rule Induction_on_int)
     with A4 I have (m+1)·D ≤ f(m·M) using Int_ZF_1_3_L13A
       by simp
  } then have ∀m∈ℤ+.(m+1)·D ≤ f(m·M) by simp
  with II show thesis by auto
qed
```

A special case of `Arthan_Lem_3` when $D = 1$.

```
corollary (in int1) Arthan_L_3_spec: assumes A1: f ∈ S+
  shows ∃M∈ℤ+.∀n∈ℤ+. n+1 ≤ f(n·M)
proof -
  have ∀n∈ℤ+. n+1 ∈ ℤ
    using PositiveSet_def int_zero_one_are_int Int_ZF_1_1_L5
    by simp
  then have ∀n∈ℤ+. (n+1)·1 = n+1
    using Int_ZF_1_1_L4 by simp
  moreover from A1 have ∃M∈ℤ+. ∀n∈ℤ+. (n+1)·1 ≤ f(n·M)
    using int_one_two_are_pos Arthan_Lem_3 by simp
  ultimately show thesis by simp
qed
```

We know from `Group_ZF_3.thy` that finite range functions are almost homomorphisms. Besides reminding that fact for slopes the next lemma shows that finite range functions do not belong to $\mathcal{S}_+$. This is important, because the projection of the set of finite range functions defines zero in the real number construction in `Real_ZF_x.thy` series, while the projection of $\mathcal{S}_+$ becomes the set of (strictly) positive reals. We don't want zero to be positive, do we? The next lemma is a part of Lemma 5 in the Arthan's paper [2].

```
lemma (in int1) Int_ZF_2_3_L1B:
  assumes A1: f ∈ FinRangeFunctions(ℤ,ℤ)
  shows f∈S   f ∉ S+
proof -
  from A1 show f∈S using Int_ZF_2_1_L1 group1.Group_ZF_3_3_L1
    by auto
  have ℤ+ ⊆ ℤ using PositiveSet_def by auto
  with A1 have f(ℤ+) ∈ Fin(ℤ)
    using Finite1_L21 by simp
  then have f(ℤ+) ∩ ℤ+ ∈ Fin(ℤ)
    using Fin_subset_lemma by blast
  thus f ∉ S+ by auto
qed
```

We want to show that if f is a slope and neither f nor $-f$ are in $\mathcal{S}_+$, then

f is bounded. The next lemma is the first step towards that goal and shows that if slope is not in $\mathcal{S}_+$ then $f(\mathbb{Z}_+)$ is bounded above.

```
lemma (in int1) Int_ZF_2_3_L2: assumes A1: f∈𝒮 and A2: f ∉ 𝒮₊
  shows IsBoundedAbove(f(ℤ₊), IntegerOrder)
proof -
  from A1 have f:ℤ→ℤ using AlmostHoms_def by simp
  then have f(ℤ₊) ⊆ ℤ using func1_1_L6 by simp
  moreover from A1 A2 have f(ℤ₊) ∩ ℤ₊ ∈ Fin(ℤ) by auto
  ultimately show thesis using Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L4
    by simp
qed
```

If f is a slope and $-f \notin \mathcal{S}_+$, then $f(\mathbb{Z}_+)$ is bounded below.

```
lemma (in int1) Int_ZF_2_3_L3: assumes A1: f∈𝒮 and A2: -f ∉ 𝒮₊
  shows IsBoundedBelow(f(ℤ₊), IntegerOrder)
proof -
  from A1 have T: f:ℤ→ℤ using AlmostHoms_def by simp
  then have (-(f(ℤ₊))) = (-f)(ℤ₊)
    using Int_ZF_1_T2 group0_2_T2 PositiveSet_def func1_1_L15C
    by auto
  with A1 A2 T show IsBoundedBelow(f(ℤ₊), IntegerOrder)
    using Int_ZF_2_1_L12 Int_ZF_2_3_L2 PositiveSet_def func1_1_L6
      Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L5 by simp
qed
```

A slope that is bounded on $\mathbb{Z}_+$ is bounded everywhere.

```
lemma (in int1) Int_ZF_2_3_L4:
  assumes A1: f∈𝒮 and A2: m∈ℤ
  and A3: ∀n∈ℤ₊. abs(f(n)) ≤ L
  shows abs(f(m)) ≤ 2·maxδ(f) + L
proof -
  from A1 A3 have
    0 ≤ abs(f(1))  abs(f(1)) ≤ L
    using int_zero_one_are_int Int_ZF_2_1_L2B int_abs_nonneg int_one_two_are_pos
    by auto
  then have II: 0≤L by (rule Int_order_transitive)
  note A2
  moreover have abs(f(0)) ≤ 2·maxδ(f) + L
  proof -
    from A1 have
      abs(f(0)) ≤ maxδ(f)  0 ≤ maxδ(f)
      and T: maxδ(f) ∈ ℤ
      using Int_ZF_2_1_L8 by auto
    with II have abs(f(0)) ≤ maxδ(f) + maxδ(f) + L
      using Int_ZF_2_L15F by simp
    with T show thesis using Int_ZF_1_1_L4 by simp
  qed
  moreover from A1 A3 II have
```

```
    ∀n∈ℤ+. abs(f(n)) ≤ 2·maxδ(f) + L
    using Int_ZF_2_1_L8 Int_ZF_1_3_L5A Int_ZF_2_L15F
    by simp
  moreover have ∀n∈ℤ+. abs(f(-n)) ≤ 2·maxδ(f) + L
  proof
    fix n assume n∈ℤ+
    with A1 A3 have
      2·maxδ(f) ∈ ℤ
      abs(f(-n)) ≤ 2·maxδ(f) + abs(f(n))
      abs(f(n)) ≤ L
      using int_two_three_are_int Int_ZF_2_1_L8 Int_ZF_1_1_L5
 PositiveSet_def Int_ZF_2_1_L14 by auto
    then show abs(f(-n)) ≤ 2·maxδ(f) + L
      using Int_ZF_2_L15A by blast
  qed
  ultimately show thesis by (rule Int_ZF_2_L19B)
qed
```

A slope whose image of the set of positive integers is bounded is a finite range function.

```
lemma (in int1) Int_ZF_2_3_L4A:
  assumes A1: f∈𝒮 and A2: IsBounded(f(ℤ+), IntegerOrder)
  shows f ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  have T1: ℤ+ ⊆ ℤ using PositiveSet_def by auto
  from A1 have T2: f:ℤ→ℤ using AlmostHoms_def by simp
  from A2 obtain L where ∀a∈f(ℤ+). abs(a) ≤ L
    using Int_ZF_1_3_L20A by auto
  with T2 T1 have ∀n∈ℤ+. abs(f(n)) ≤ L
    by (rule func1_1_L15B)
  with A1 have ∀m∈ℤ. abs(f(m)) ≤ 2·maxδ(f) + L
    using Int_ZF_2_3_L4 by simp
  with T2 have f(ℤ) ∈ Fin(ℤ)
    by (rule Int_ZF_1_3_L20C)
  with T2 show f ∈ FinRangeFunctions(ℤ,ℤ)
    using FinRangeFunctions_def by simp
qed
```

A slope whose image of the set of positive integers is bounded below is a finite range function or a positive slope.

```
lemma (in int1) Int_ZF_2_3_L4B:
  assumes f∈𝒮 and IsBoundedBelow(f(ℤ+), IntegerOrder)
  shows f ∈ FinRangeFunctions(ℤ,ℤ) ∨ f∈𝒮+
  using assms Int_ZF_2_3_L2 IsBounded_def Int_ZF_2_3_L4A
  by auto
```

If one slope is not greater then another on positive integers, then they are almost equal or the difference is a positive slope.

```
lemma (in int1) Int_ZF_2_3_L4C: assumes A1: f∈𝒮  g∈𝒮 and
```

```
  A2: ∀n∈ℤ+. f(n) ≤ g(n)
  shows f∼g ∨ g + (-f) ∈ 𝒮+
proof -
  let h = g + (-f)
  from A1 have (-f) ∈ 𝒮 using Int_ZF_2_1_L12
    by simp
  with A1 have I: h ∈ 𝒮 using Int_ZF_2_1_L12C
    by simp
  moreover have IsBoundedBelow(h(ℤ+), IntegerOrder)
  proof -
    from I have
      h:ℤ→ℤ and ℤ+⊆ℤ using AlmostHoms_def PositiveSet_def
      by auto
    moreover from A1 A2 have ∀n∈ℤ+. ⟨0, h(n)⟩ ∈ IntegerOrder
      using Int_ZF_2_1_L2B PositiveSet_def Int_ZF_1_3_L10A
 Int_ZF_2_1_L12 Int_ZF_2_1_L12B Int_ZF_2_1_L12A
      by simp
    ultimately show IsBoundedBelow(h(ℤ+), IntegerOrder)
      by (rule func_ZF_8_L1)
  qed
  ultimately have h ∈ FinRangeFunctions(ℤ,ℤ) ∨ h∈𝒮+
    using Int_ZF_2_3_L4B by simp
  with A1 show f∼g ∨ g + (-f) ∈ 𝒮+
    using Int_ZF_2_1_L9C by auto
qed
```

Positive slopes are arbitrarily large for large enough arguments.

```
lemma (in int1) Int_ZF_2_3_L5:
  assumes A1: f∈𝒮+ and A2: K∈ℤ
  shows ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
proof -
  from A1 obtain M where I: M∈ℤ+ and II: ∀n∈ℤ+. n+1 ≤ f(n·M)
    using Arthan_L_3_spec by auto
  let j = GreaterOf(IntegerOrder,M,K - (minf(f,0..(M-1)) - maxδ(f)) -
1)
  from A1 I have T1:
    minf(f,0..(M-1)) - maxδ(f) ∈ ℤ  M∈ℤ
    using Int_ZF_2_1_L15 Int_ZF_2_1_L8 Int_ZF_1_1_L5 PositiveSet_def
    by auto
  with A2 I have T2:
    K - (minf(f,0..(M-1)) - maxδ(f)) ∈ ℤ
    K - (minf(f,0..(M-1)) - maxδ(f)) - 1 ∈ ℤ
    using Int_ZF_1_1_L5 int_zero_one_are_int by auto
  with T1 have III: M ≤ j  and
    K - (minf(f,0..(M-1)) - maxδ(f)) - 1 ≤ j
    using Int_ZF_1_3_L18 by auto
  with A2 T1 T2 have
    IV: K ≤ j+1 + (minf(f,0..(M-1)) - maxδ(f))
    using int_zero_one_are_int Int_ZF_2_L9C by simp
```

```
  let N = GreaterOf(IntegerOrder,1,j·M)
  from T1 III have T3: j ∈ ℤ  j·M ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  then have V: N ∈ ℤ₊ and VI: j·M ≤ N
    using int_zero_one_are_int Int_ZF_1_5_L3 Int_ZF_1_3_L18
    by auto
  { fix m
    let n = m zdiv M
    let k = m zmod M
    assume N≤m
    with VI have j·M ≤ m by (rule Int_order_transitive)
    with I III have
      VII: m = n·M+k
      j ≤ n  and
      VIII: n ∈ ℤ₊  k ∈ 0..(M-1)
      using IntDiv_ZF_1_L5 by auto
    with II have
      j + 1 ≤ n + 1  n+1 ≤ f(n·M)
      using int_zero_one_are_int int_ord_transl_inv by auto
    then have j + 1 ≤  f(n·M)
      by (rule Int_order_transitive)
    with T1 have
      j+1 + (minf(f,0..(M-1)) - maxδ(f)) ≤
      f(n·M) + (minf(f,0..(M-1)) - maxδ(f))
      using int_ord_transl_inv by simp
    with IV have K ≤ f(n·M) + (minf(f,0..(M-1)) - maxδ(f))
      by (rule Int_order_transitive)
    moreover from A1 I VIII have
      f(n·M) + (minf(f,0..(M-1))- maxδ(f)) ≤ f(n·M+k)
      using PositiveSet_def Int_ZF_2_1_L16 by simp
    ultimately have K ≤ f(n·M+k)
      by (rule Int_order_transitive)
    with VII have K ≤ f(m) by simp
    } then have  ∀m. N≤m ⟶ K ≤ f(m)
      by simp
    with V show thesis by auto
qed
```

Positive slopes are arbitrarily small for small enough arguments. Kind of dual to `Int_ZF_2_3_L5`.

```
lemma (in int1) Int_ZF_2_3_L5A: assumes A1: f∈𝒮₊ and A2: K∈ℤ
  shows ∃N∈ℤ₊. ∀m. N≤m ⟶ f(-m) ≤ K
proof -
  from A1 have T1: abs(f(0)) + maxδ(f) ∈ ℤ
    using Int_ZF_2_1_L8 by auto
  with A2 have abs(f(0)) + maxδ(f) - K ∈ ℤ
    using Int_ZF_1_1_L5 by simp
  with A1 have
    ∃N∈ℤ₊. ∀m. N≤m ⟶ abs(f(0)) + maxδ(f) - K ≤ f(m)
```

```
    using Int_ZF_2_3_L5 by simp
  then obtain N where I: N∈ℤ₊ and II:
    ∀m. N≤m ⟶  abs(f(0)) + maxδ(f) - K ≤ f(m)
    by auto
  { fix m assume A3: N≤m
    with A1 have
      f(-m) ≤ abs(f(0)) + maxδ(f) - f(m)
      using Int_ZF_2_L1A Int_ZF_2_1_L14 by simp
    moreover
    from II T1 A3 have abs(f(0)) + maxδ(f) - f(m) ≤
      (abs(f(0)) + maxδ(f)) -(abs(f(0)) + maxδ(f) - K)
      using Int_ZF_2_L10 int_ord_transl_inv by simp
    with A2 T1 have abs(f(0)) + maxδ(f) - f(m) ≤ K
      using Int_ZF_1_2_L3 by simp
    ultimately have f(-m) ≤ K
      by (rule Int_order_transitive)
  } then have ∀m. N≤m  ⟶ f(-m) ≤ K
    by simp
  with I show thesis by auto
qed
```

A special case of `Int_ZF_2_3_L5` where $K = 1$.

```
corollary (in int1) Int_ZF_2_3_L6: assumes f∈𝒮₊
  shows ∃N∈ℤ₊. ∀m. N≤m ⟶ f(m) ∈ ℤ₊
  using assms int_zero_one_are_int Int_ZF_2_3_L5 Int_ZF_1_5_L3
  by simp
```

A special case of `Int_ZF_2_3_L5` where $m = N$.

```
corollary (in int1) Int_ZF_2_3_L6A: assumes f∈𝒮₊ and K∈ℤ
   shows ∃N∈ℤ₊. K ≤ f(N)
proof -
  from assms have ∃N∈ℤ₊. ∀m. N≤m ⟶ K ≤ f(m)
    using Int_ZF_2_3_L5 by simp
  then obtain N where I: N ∈ ℤ₊  and II: ∀m. N≤m ⟶ K ≤ f(m)
    by auto
  then show thesis using PositiveSet_def int_ord_is_refl refl_def
    by auto
qed
```

If values of a slope are not bounded above, then the slope is positive.

```
lemma (in int1) Int_ZF_2_3_L7: assumes A1: f∈𝒮
  and A2: ∀K∈ℤ. ∃n∈ℤ₊. K ≤ f(n)
  shows f ∈ 𝒮₊
proof -
  { fix K assume K∈ℤ
    with A2 obtain n where n∈ℤ₊  K ≤ f(n)
      by auto
    moreover from A1 have ℤ₊ ⊆ ℤ  f:ℤ→ℤ
      using PositiveSet_def AlmostHoms_def by auto
```

```
    ultimately have ∃m ∈ f(ℤ+). K ≤ m
      using func1_1_L15D by auto
  } then have ∀K∈ℤ. ∃m ∈ f(ℤ+). K ≤ m by simp
  with A1 show f ∈ 𝒮+ using Int_ZF_4_L9 Int_ZF_2_3_L2
    by auto
qed
```

For unbounded slope f either $f \in\mathcal{S}_+$ of $-f \in\mathcal{S}_+$.

```
theorem (in int1) Int_ZF_2_3_L8:
  assumes A1: f∈𝒮 and A2: f ∉ FinRangeFunctions(ℤ,ℤ)
  shows (f ∈ 𝒮+) Xor ((-f) ∈ 𝒮+)
proof -
  have T1: ℤ+ ⊆ ℤ using PositiveSet_def by auto
  from A1 have T2: f:ℤ→ℤ  using AlmostHoms_def by simp
  then have I: f(ℤ+) ⊆ ℤ using func1_1_L6 by auto
  from A1 A2 have f ∈ 𝒮+ ∨ (-f) ∈ 𝒮+
    using Int_ZF_2_3_L2 Int_ZF_2_3_L3 IsBounded_def Int_ZF_2_3_L4A
    by blast
  moreover have ¬(f ∈ 𝒮+ ∧ (-f) ∈ 𝒮+)
  proof -
    { assume A3: f ∈ 𝒮+  and A4: (-f) ∈ 𝒮+
      from A3 obtain N1 where
 I: N1∈ℤ+ and II: ∀m. N1≤m ⟶ f(m) ∈ ℤ+
 using Int_ZF_2_3_L6 by auto
      from A4 obtain N2 where
 III: N2∈ℤ+ and IV: ∀m. N2≤m ⟶ (-f)(m) ∈ ℤ+
 using Int_ZF_2_3_L6 by auto
      let N = GreaterOf(IntegerOrder,N1,N2)
      from I III have N1 ≤ N  N2 ≤ N
 using PositiveSet_def Int_ZF_1_3_L18 by auto
      with A1 II IV have
 f(N) ∈ ℤ+  (-f)(N) ∈ ℤ+  (-f)(N) = -(f(N))
 using Int_ZF_2_L1A PositiveSet_def Int_ZF_2_1_L12A
 by auto
      then have False using Int_ZF_1_5_L8 by simp
    } thus thesis by auto
  qed
  ultimately show (f ∈ 𝒮+) Xor ((-f) ∈ 𝒮+)
    using Xor_def by simp
qed
```

The sum of positive slopes is a positive slope.

```
theorem (in int1) sum_of_pos_sls_is_pos_sl:
  assumes A1: f ∈ 𝒮+  g ∈ 𝒮+
  shows f+g ∈ 𝒮+
proof -
  { fix K assume K∈ℤ
    with A1 have ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
      using Int_ZF_2_3_L5 by simp
```

```
    then obtain N where I: N∈ℤ+ and II: ∀m. N≤m ⟶ K ≤ f(m)
      by auto
    from A1 have ∃M∈ℤ+. ∀m. M≤m ⟶ 0 ≤ g(m)
      using int_zero_one_are_int Int_ZF_2_3_L5 by simp
    then obtain M where III: M∈ℤ+ and IV: ∀m. M≤m ⟶ 0 ≤ g(m)
      by auto
    let L = GreaterOf(IntegerOrder,N,M)
    from I III have V: L ∈ ℤ+  ℤ+ ⊆ ℤ
      using GreaterOf_def PositiveSet_def by auto
    moreover from A1 V have (f+g)(L) = f(L) + g(L)
      using Int_ZF_2_1_L12B by auto
    moreover from I II III IV have K ≤ f(L) + g(L)
      using PositiveSet_def Int_ZF_1_3_L18 Int_ZF_2_L15F
      by simp
    ultimately have L ∈ ℤ+  K ≤ (f+g)(L)
      by auto
    then have ∃n ∈ℤ+. K ≤ (f+g)(n)
      by auto
  } with A1 show f+g ∈ 𝒮+
    using Int_ZF_2_1_L12C Int_ZF_2_3_L7 by simp
qed
```

The composition of positive slopes is a positive slope.

```
theorem (in int1) comp_of_pos_sls_is_pos_sl:
  assumes A1: f ∈ 𝒮+  g ∈ 𝒮+
  shows f∘g ∈ 𝒮+
proof -
  { fix K assume K∈ℤ
    with A1 have ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
      using Int_ZF_2_3_L5 by simp
    then obtain N where N∈ℤ+ and I: ∀m. N≤m ⟶ K ≤ f(m)
      by auto
    with A1 have ∃M∈ℤ+. N ≤ g(M)
      using PositiveSet_def Int_ZF_2_3_L6A by simp
    then obtain M where M∈ℤ+  N ≤ g(M)
      by auto
    with A1 I have ∃M∈ℤ+. K ≤ (f∘g)(M)
      using PositiveSet_def Int_ZF_2_1_L10
      by auto
  } with A1 show f∘g ∈ 𝒮+
    using Int_ZF_2_1_L11 Int_ZF_2_3_L7
    by simp
qed
```

A slope equivalent to a positive one is positive.

```
lemma (in int1) Int_ZF_2_3_L9:
  assumes A1: f ∈ 𝒮+ and A2: ⟨f,g⟩ ∈ AlEqRel shows g ∈ 𝒮+
proof -
  from A2 have T: g∈𝒮 and ∃L∈ℤ. ∀m∈ℤ. abs(f(m)-g(m)) ≤ L
```

```
    using Int_ZF_2_1_L9A by auto
   then obtain L where
     I: L∈ℤ  and II: ∀m∈ℤ. abs(f(m)-g(m)) ≤ L
     by auto
  { fix K assume A3: K∈ℤ
    with I have K+L ∈ ℤ
      using Int_ZF_1_1_L5 by simp
    with A1 obtain M where III: M∈ℤ₊  and IV: K+L ≤ f(M)
      using Int_ZF_2_3_L6A by auto
    with A1 A3 I have  K ≤ f(M)-L
      using PositiveSet_def Int_ZF_2_1_L2B Int_ZF_2_L9B
      by simp
    moreover from A1 T II III have
      f(M)-L ≤ g(M)
      using PositiveSet_def Int_ZF_2_1_L2B Int_triangle_ineq2
      by simp
    ultimately have K ≤  g(M)
      by (rule Int_order_transitive)
    with III have ∃n∈ℤ₊. K ≤ g(n)
      by auto
  } with T show g ∈ 𝒮₊
    using Int_ZF_2_3_L7 by simp
qed
```

The set of positive slopes is saturated with respect to the relation of equivalence of slopes.

```
lemma (in int1) pos_slopes_saturated: shows IsSaturated(AlEqRel,𝒮₊)
proof -
  have
    equiv(𝒮,AlEqRel)
    AlEqRel ⊆ 𝒮 × 𝒮
    using Int_ZF_2_1_L9B by auto
  moreover have 𝒮₊ ⊆ 𝒮 by auto
  moreover have ∀f∈𝒮₊. ∀g∈𝒮. ⟨f,g⟩ ∈ AlEqRel ⟶ g ∈ 𝒮₊
    using Int_ZF_2_3_L9 by blast
  ultimately show IsSaturated(AlEqRel,𝒮₊)
    by (rule EquivClass_3_L3)
qed
```

A technical lemma involving a projection of the set of positive slopes and a logical epression with exclusive or.

```
lemma (in int1) Int_ZF_2_3_L10:
  assumes A1: f∈𝒮  g∈𝒮
  and A2: R = {AlEqRel{s}. s∈𝒮₊}
  and A3: (f∈𝒮₊) Xor (g∈𝒮₊)
  shows (AlEqRel{f} ∈ R) Xor (AlEqRel{g} ∈ R)
proof -
  from A1 A2 A3 have
    equiv(𝒮,AlEqRel)
```

```
    IsSaturated(AlEqRel,𝒮₊)
    𝒮₊ ⊆ 𝒮
    f∈𝒮  g∈𝒮
    R = {AlEqRel{s}. s∈𝒮₊}
    (f∈𝒮₊) Xor (g∈𝒮₊)
    using pos_slopes_saturated Int_ZF_2_1_L9B by auto
  then show thesis by (rule EquivClass_3_L7)
qed
```

Identity function is a positive slope.

```
lemma (in int1) Int_ZF_2_3_L11: shows id(ℤ) ∈ 𝒮₊
proof -
  let f = id(ℤ)
  { fix K assume K∈ℤ
    then obtain n where T: n∈ℤ₊ and K≤n
      using Int_ZF_1_5_L9 by auto
    moreover from T have f(n) = n
      using PositiveSet_def by simp
    ultimately have  n∈ℤ₊ and K≤f(n)
      by auto
    then have ∃n∈ℤ₊. K≤f(n) by auto
  } then show f ∈ 𝒮₊
    using Int_ZF_2_1_L17 Int_ZF_2_3_L7 by simp
qed
```

The identity function is not almost equal to any bounded function.

```
lemma (in int1) Int_ZF_2_3_L12: assumes A1: f ∈ FinRangeFunctions(ℤ,ℤ)
  shows ¬(id(ℤ) ∼ f)
proof -
  { from A1 have id(ℤ) ∈ 𝒮₊
      using Int_ZF_2_3_L11 by simp
    moreover assume ⟨id(ℤ),f⟩ ∈ AlEqRel
    ultimately have  f ∈ 𝒮₊
      by (rule Int_ZF_2_3_L9)
    with A1 have False using Int_ZF_2_3_L1B
      by simp
  } then show ¬(id(ℤ) ∼ f) by auto
qed
```

59.2 Inverting slopes

Not every slope is a 1:1 function. However, we can still invert slopes in the sense that if f is a slope, then we can find a slope g such that $f \circ g$ is almost equal to the identity function. The goal of this this section is to establish this fact for positive slopes.

If f is a positive slope, then for every positive integer p the set $\{n \in Z_+ : p \leq f(n)\}$ is a nonempty subset of positive integers. Recall that $f^{-1}(p)$ is the notation for the smallest element of this set.

```
lemma (in int1) Int_ZF_2_4_L1:
  assumes A1: f ∈ 𝒮+ and A2: p∈ℤ+ and A3: A = {n∈ℤ+. p ≤ f(n)}
  shows
  A ⊆ ℤ+
  A ≠ 0
  f⁻¹(p) ∈ A
  ∀m∈A. f⁻¹(p) ≤ m
proof -
  from A3 show I: A ⊆ ℤ+ by auto
  from A1 A2 have ∃n∈ℤ+. p ≤ f(n)
    using PositiveSet_def Int_ZF_2_3_L6A by simp
  with A3 show II: A ≠ 0 by auto
  from A3 I II show
    f⁻¹(p) ∈ A
    ∀m∈A. f⁻¹(p) ≤ m
    using Int_ZF_1_5_L1C by auto
qed
```

If f is a positive slope and p is a positive integer p, then $f^{-1}(p)$ (defined as the minimum of the set $\{n \in Z_+ : p \leq f(n)\}$) is a (well defined) positive integer.

```
lemma (in int1) Int_ZF_2_4_L2:
  assumes f ∈ 𝒮+ and p∈ℤ+
  shows
  f⁻¹(p) ∈ ℤ+
  p ≤ f(f⁻¹(p))
  using assms Int_ZF_2_4_L1 by auto
```

If f is a positive slope and p is a positive integer such that $n \leq f(p)$, then $f^{-1}(n) \leq p$.

```
lemma (in int1) Int_ZF_2_4_L3:
  assumes f ∈ 𝒮+ and  m∈ℤ+  p∈ℤ+ and m ≤ f(p)
  shows f⁻¹(m) ≤ p
  using assms Int_ZF_2_4_L1 by simp
```

An upper bound $f(f^{-1}(m) - 1)$ for positive slopes.

```
lemma (in int1) Int_ZF_2_4_L4:
  assumes A1: f ∈ 𝒮+ and A2: m∈ℤ+ and A3: f⁻¹(m)-1 ∈ ℤ+
  shows f(f⁻¹(m)-1) ≤ m   f(f⁻¹(m)-1) ≠ m
proof -
  from A1 A2 have T: f⁻¹(m) ∈ ℤ using Int_ZF_2_4_L2 PositiveSet_def
    by simp
  from A1 A3 have f:ℤ→ℤ  and f⁻¹(m)-1 ∈ ℤ
    using Int_ZF_2_3_L1 PositiveSet_def by auto
  with A1 A2 have T1: f(f⁻¹(m)-1) ∈ ℤ  m∈ℤ
    using apply_funtype PositiveSet_def by auto
  { assume m ≤ f(f⁻¹(m)-1)
    with A1 A2 A3 have f⁻¹(m) ≤ f⁻¹(m)-1
```

```
      by (rule Int_ZF_2_4_L3)
    with T have False using Int_ZF_1_2_L3AA
      by simp
  } then have I: ¬(m ≤ f(f⁻¹(m)-1)) by auto
  with T1 show f(f⁻¹(m)-1) ≤ m
    by (rule Int_ZF_2_L19)
  from T1 I show f(f⁻¹(m)-1) ≠ m
    by (rule Int_ZF_2_L19)
qed
```

The (candidate for) the inverse of a positive slope is nondecreasing.

```
lemma (in int1) Int_ZF_2_4_L5:
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊ and A3: m≤n
  shows f⁻¹(m) ≤ f⁻¹(n)
proof -
  from A2 A3 have T: n ∈ ℤ₊ using Int_ZF_1_5_L7 by blast
  with A1 have n ≤ f(f⁻¹(n)) using Int_ZF_2_4_L2
    by simp
  with A3 have m ≤ f(f⁻¹(n)) by (rule Int_order_transitive)
  with A1 A2 T show f⁻¹(m) ≤ f⁻¹(n)
    using Int_ZF_2_4_L2 Int_ZF_2_4_L3 by simp
qed
```

If $f^{-1}(m)$ is positive and n is a positive integer, then, then $f^{-1}(m+n)-1$ is positive.

```
lemma (in int1) Int_ZF_2_4_L6:
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊  n∈ℤ₊ and
  A3: f⁻¹(m)-1 ∈ ℤ₊
  shows f⁻¹(m+n)-1 ∈ ℤ₊
proof -
  from A1 A2 have f⁻¹(m)-1 ≤  f⁻¹(m+n) - 1
     using PositiveSet_def Int_ZF_1_5_L7A Int_ZF_2_4_L2
       Int_ZF_2_4_L5 int_zero_one_are_int Int_ZF_1_1_L4
       int_ord_transl_inv by simp
  with A3 show f⁻¹(m+n)-1 ∈ ℤ₊ using Int_ZF_1_5_L7
    by blast
qed
```

If f is a slope, then $f(f^{-1}(m+n)-f^{-1}(m)-f^{-1}(n))$ is uniformly bounded above and below. Will it be the messiest IsarMathLib proof ever? Only time will tell.

```
lemma (in int1) Int_ZF_2_4_L7:  assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-1 ∈ ℤ₊
  shows
  ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
  ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
proof -
  from A1 have ∃L∈ℤ. ∀r∈ℤ. f(r) ≤ f(r-1) + L
```

```
    using Int_ZF_2_1_L28 by simp
  then obtain L where
    I: L∈ℤ and II: ∀r∈ℤ. f(r) ≤ f(r-1) + L
    by auto
  from A1 have
    ∃M∈ℤ. ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    ∃K∈ℤ. ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r)-f(p)-f(q)+K ≤ f(r-p-q)
    using Int_ZF_2_1_L30 by auto
  then obtain M K where III: M∈ℤ and
    IV: ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    and
    V: K∈ℤ and VI: ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r)-f(p)-f(q)+K ≤ f(r-p-q)
    by auto
  from I III V have
    L+M ∈ ℤ  (-L) - L + K ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_1_1_L5 by auto
  moreover
    { fix m n
      assume A3: m∈ℤ₊ n∈ℤ₊
      have  f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤  L+M ∧
(-L)-L+K ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
      proof -
let r = f⁻¹(m+n)
let p = f⁻¹(m)
let q = f⁻¹(n)
from A1 A3 have T1:
  p ∈ ℤ₊  q ∈ ℤ₊  r ∈ ℤ₊
  using Int_ZF_2_4_L2 pos_int_closed_add_unfolded by auto
with A3 have T2:
  m ∈ ℤ  n ∈ ℤ  p ∈ ℤ  q ∈ ℤ  r ∈ ℤ
  using PositiveSet_def by auto
from A2 A3 have T3:
  r-1 ∈ ℤ₊ p-1 ∈ ℤ₊  q-1 ∈ ℤ₊
  using pos_int_closed_add_unfolded by auto
from A1 A3 have VII:
  m+n ≤ f(r)
  m ≤ f(p)
  n ≤ f(q)
  using Int_ZF_2_4_L2 pos_int_closed_add_unfolded by auto
from A1 A3 T3 have VIII:
  f(r-1) ≤ m+n
  f(p-1) ≤ m
  f(q-1) ≤ n
  using pos_int_closed_add_unfolded Int_ZF_2_4_L4 by auto
have f(r-p-q) ≤ L+M
proof -
  from IV T2 have f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    by simp
  moreover
```

```
  from I II T2 VIII have
    f(r) ≤ f(r-1) + L
    f(r-1) + L ≤ m+n+L
    using int_ord_transl_inv by auto
  then have f(r) ≤  m+n+L
    by (rule Int_order_transitive)
  with VII have f(r) - f(p) ≤ m+n+L-m
    using int_ineq_add_sides by simp
  with I T2 VII have f(r) - f(p) - f(q) ≤ n+L-n
    using Int_ZF_1_2_L9 int_ineq_add_sides by simp
  with I III T2 have f(r) - f(p) - f(q) + M ≤ L+M
    using Int_ZF_1_2_L3 int_ord_transl_inv by simp
  ultimately show f(r-p-q) ≤ L+M
    by (rule Int_order_transitive)
qed
moreover have (-L)-L +K ≤ f(r-p-q)
proof -
  from I II T2 VIII have
    f(p) ≤ f(p-1) + L
    f(p-1) + L ≤ m +L
    using int_ord_transl_inv by auto
  then have f(p) ≤  m +L
    by (rule Int_order_transitive)
  with VII have m+n -(m+L) ≤ f(r) - f(p)
    using int_ineq_add_sides by simp
  with I T2 have n - L ≤  f(r) - f(p)
    using Int_ZF_1_2_L9 by simp
  moreover
  from I II T2 VIII have
    f(q) ≤ f(q-1) + L
    f(q-1) + L ≤ n +L
    using int_ord_transl_inv by auto
  then have f(q) ≤  n +L
    by (rule Int_order_transitive)
  ultimately have
    n - L - (n+L) ≤  f(r) - f(p) - f(q)
    using int_ineq_add_sides by simp
  with I V T2 have
    (-L)-L +K ≤  f(r) - f(p) - f(q) + K
    using Int_ZF_1_2_L3 int_ord_transl_inv by simp
  moreover from VI T2 have
    f(r) - f(p) - f(q) + K ≤ f(r-p-q)
    by simp
  ultimately show (-L)-L +K ≤ f(r-p-q)
    by (rule Int_order_transitive)
qed
ultimately show
  f(r-p-q) ≤  L+M ∧
  (-L)-L+K ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
```

```
    by simp
       qed
     }
  ultimately show
    ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    by auto
qed
```

The expression $f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$ is uniformly bounded for all pairs $\langle m,n\rangle \in \mathbb{Z}_+ \times \mathbb{Z}_+$. Recall that in the `int1` context `ε(f,x)` is defined so that $\varepsilon(f,\langle m,n\rangle) = f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$.

```
lemma (in int1) Int_ZF_2_4_L8:  assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-1 ∈ ℤ₊
  shows ∃M. ∀x∈ℤ₊×ℤ₊. abs(ε(f,x)) ≤ M
proof -
  from A1 A2 have
    ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    using  Int_ZF_2_4_L7 by auto
  then obtain U N where I:
    ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    by auto
  have ℤ₊×ℤ₊ ≠ 0 using int_one_two_are_pos by auto
  moreover from A1 have f: ℤ→ℤ
    using AlmostHoms_def by simp
  moreover from A1 have
    ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x)
    using Int_ZF_2_3_L5 by simp
  moreover from A1 have
    ∀a∈ℤ.∃b∈ℤ₊.∀y. b≤y ⟶ f(-y) ≤ a
    using Int_ZF_2_3_L5A by simp
  moreover have
    ∀x∈ℤ₊×ℤ₊. ε(f,x) ∈ ℤ ∧ f(ε(f,x)) ≤ U ∧ N ≤ f(ε(f,x))
  proof -
    { fix x assume A3: x ∈ ℤ₊×ℤ₊
      let m = fst(x)
      let n = snd(x)
      from A3 have T: m ∈ ℤ₊  n ∈ ℤ₊  m+n ∈ ℤ₊
 using pos_int_closed_add_unfolded by auto
      with A1 have
 f⁻¹(m+n) ∈ ℤ  f⁻¹(m) ∈ ℤ  f⁻¹(n) ∈ ℤ
 using Int_ZF_2_4_L2 PositiveSet_def by auto
      with I T have
 ε(f,x) ∈ ℤ ∧ f(ε(f,x)) ≤ U ∧ N ≤ f(ε(f,x))
 using Int_ZF_1_1_L5 by auto
    } thus thesis by simp
    qed
```

```
  ultimately show ∃M.∀x∈ℤ₊×ℤ₊. abs(ε(f,x)) ≤ M
    by (rule Int_ZF_1_6_L4)
qed
```

The (candidate for) inverse of a positive slope is a (well defined) function on $\mathbb{Z}_+$.

```
lemma (in int1) Int_ZF_2_4_L9:
  assumes A1: f ∈ 𝒮₊ and A2: g = {⟨p,f⁻¹(p)⟩. p∈ℤ₊}
  shows
  g : ℤ₊→ℤ₊
  g : ℤ₊→ℤ
proof -
  from A1 have
    ∀p∈ℤ₊. f⁻¹(p) ∈ ℤ₊
    ∀p∈ℤ₊. f⁻¹(p) ∈ ℤ
    using Int_ZF_2_4_L2 PositiveSet_def by auto
  with A2 show
    g : ℤ₊→ℤ₊  and  g : ℤ₊→ℤ
    using ZF_fun_from_total by auto
qed
```

What are the values of the (candidate for) the inverse of a positive slope?

```
lemma (in int1) Int_ZF_2_4_L10:
  assumes A1: f ∈ 𝒮₊ and A2: g = {⟨p,f⁻¹(p)⟩. p∈ℤ₊} and A3: p∈ℤ₊
  shows g(p) = f⁻¹(p)
proof -
  from A1 A2 have  g : ℤ₊→ℤ₊ using Int_ZF_2_4_L9 by simp
  with A2 A3 show g(p) = f⁻¹(p) using ZF_fun_from_tot_val by simp
qed
```

The (candidate for) the inverse of a positive slope is a slope.

```
lemma (in int1) Int_ZF_2_4_L11: assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-1 ∈ ℤ₊ and
  A3: g = {⟨p,f⁻¹(p)⟩. p∈ℤ₊}
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,g) ∈ 𝒮
proof -
  from A1 A2 have ∃L. ∀x∈ℤ₊×ℤ₊. abs(ε(f,x)) ≤ L
    using Int_ZF_2_4_L8 by simp
  then obtain L where I: ∀x∈ℤ₊×ℤ₊. abs(ε(f,x)) ≤ L
    by auto
  from A1 A3 have g : ℤ₊→ℤ using Int_ZF_2_4_L9
    by simp
  moreover have ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ L
  proof-
    { fix m n
      assume A4: m∈ℤ₊  n∈ℤ₊
      then have ⟨m,n⟩ ∈ ℤ₊×ℤ₊ by simp
      with I have abs(ε(f,⟨m,n⟩)) ≤ L by simp
      moreover have ε(f,⟨m,n⟩) = f⁻¹(m+n) - f⁻¹(m) - f⁻¹(n)
```

```
 by simp
      moreover from A1 A3 A4 have
 f⁻¹(m+n) = g(m+n)  f⁻¹(m) = g(m)  f⁻¹(n) = g(n)
 using pos_int_closed_add_unfolded Int_ZF_2_4_L10 by auto
      ultimately have abs(δ(g,m,n)) ≤ L by simp
    } thus ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ L by simp
  qed
  ultimately show thesis by (rule Int_ZF_2_1_L24)
qed
```

Every positive slope that is at least 2 on positive integers almost has an inverse.

```
lemma (in int1) Int_ZF_2_4_L12: assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-1 ∈ ℤ₊
  shows ∃h∈𝒮. f∘h ∼ id(ℤ)
proof -
  let g = {⟨p,f⁻¹(p)⟩. p∈ℤ₊}
  let h = OddExtension(ℤ,IntegerAddition,IntegerOrder,g)
  from A1 have
    ∃M∈ℤ. ∀n∈ℤ. f(n) ≤ f(n-1) + M
    using Int_ZF_2_1_L28 by simp
  then obtain M where
    I: M∈ℤ and II: ∀n∈ℤ. f(n) ≤ f(n-1) + M
    by auto
  from A1 A2 have T: h ∈ 𝒮
    using Int_ZF_2_4_L11 by simp
  moreover have  f∘h ∼ id(ℤ)
  proof -
    from A1 T have f∘h ∈ 𝒮 using Int_ZF_2_1_L11
      by simp
    moreover note I
    moreover
    { fix m assume A3: m∈ℤ₊
      with A1 have f⁻¹(m) ∈ ℤ
 using Int_ZF_2_4_L2 PositiveSet_def by simp
      with II have f(f⁻¹(m)) ≤ f(f⁻¹(m)-1) + M
 by simp
      moreover from A1 A2 I A3 have f(f⁻¹(m)-1) + M ≤ m+M
 using Int_ZF_2_4_L4 int_ord_transl_inv by simp
      ultimately have f(f⁻¹(m)) ≤ m+M
 by (rule Int_order_transitive)
      moreover from A1 A3 have m ≤ f(f⁻¹(m))
 using Int_ZF_2_4_L2 by simp
      moreover from A1 A2 T A3 have f(f⁻¹(m)) = (f∘h)(m)
 using Int_ZF_2_4_L9 Int_ZF_1_5_L11
   Int_ZF_2_4_L10 PositiveSet_def Int_ZF_2_1_L10
 by simp
      ultimately have m ≤ (f∘h)(m) ∧ (f∘h)(m) ≤ m+M
 by simp }
```

```
    ultimately show f∘h ∼ id(ℤ) using Int_ZF_2_1_L32
      by simp
  qed
  ultimately show ∃h∈𝒮. f∘h ∼ id(ℤ)
    by auto
qed
```

Int_ZF_2_4_L12 is almost what we need, except that it has an assumption that the values of the slope that we get the inverse for are not smaller than 2 on positive integers. The Arthan's proof of Theorem 11 has a mistake where he says "note that for all but finitely many $m,n \in N$ $p = g(m)$ and $q = g(n)$ are both positive". Of course there may be infinitely many pairs $\langle m,n\rangle$ such that p,q are not both positive. This is however easy to workaround: we just modify the slope by adding a constant so that the slope is large enough on positive integers and then look for the inverse.

```
theorem (in int1) pos_slope_has_inv: assumes A1: f ∈ 𝒮₊
  shows ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(ℤ))
proof -
  from A1 have f: ℤ→ℤ  1∈ℤ  2 ∈ ℤ
    using AlmostHoms_def int_zero_one_are_int int_two_three_are_int
    by auto
  moreover from A1 have
     ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x)
    using Int_ZF_2_3_L5 by simp
  ultimately have
    ∃c∈ℤ. 2 ≤ Minimum(IntegerOrder,{n∈ℤ₊. 1 ≤ f(n)+c})
    by (rule Int_ZF_1_6_L7)
  then obtain c where I: c∈ℤ and
    II: 2 ≤ Minimum(IntegerOrder,{n∈ℤ₊. 1 ≤ f(n)+c})
    by auto
  let g = {⟨m,f(m)+c⟩. m∈ℤ}
  from A1 I have III: g∈𝒮 and IV: f∼g using Int_ZF_2_1_L33
    by auto
  from IV have ⟨f,g⟩ ∈ AlEqRel by simp
  with A1 have T: g ∈ 𝒮₊ by (rule Int_ZF_2_3_L9)
  moreover have ∀m∈ℤ₊. g⁻¹(m)-1 ∈ ℤ₊
  proof
    fix m assume A2: m∈ℤ₊
    from A1 I II have V: 2 ≤ g⁻¹(1)
      using Int_ZF_2_1_L33 PositiveSet_def by simp
    moreover from A2 T have g⁻¹(1) ≤ g⁻¹(m)
      using Int_ZF_1_5_L3 int_one_two_are_pos Int_ZF_2_4_L5
      by simp
    ultimately have 2 ≤ g⁻¹(m)
      by (rule Int_order_transitive)
    then have 2-1 ≤ g⁻¹(m)-1
      using int_zero_one_are_int Int_ZF_1_1_L4 int_ord_transl_inv
      by simp
```

```
    then show  g⁻¹(m)-1 ∈ ℤ₊
      using int_zero_one_are_int Int_ZF_1_2_L3 Int_ZF_1_5_L3
      by simp
  qed
  ultimately have ∃h∈𝒮. g∘h ∼ id(ℤ)
    by (rule Int_ZF_2_4_L12)
  with III IV show thesis by auto
qed
```

59.3 Completeness

In this section we consider properties of slopes that are needed for the proof of completeness of real numbers constructred in `Real_ZF_1.thy`. In particular we consider properties of embedding of integers into the set of slopes by the mapping $m \mapsto m^S$, where m^S is defined by $m^S(n) = m \cdot n$.

If m is an integer, then m^S is a slope whose value is $m \cdot n$ for every integer.

```
lemma (in int1) Int_ZF_2_5_L1: assumes A1: m ∈ ℤ
  shows
  ∀n ∈ ℤ. (m^S)(n) = m·n
  m^S ∈ 𝒮
proof -
  from A1 have I: m^S:ℤ→ℤ
    using Int_ZF_1_1_L5 ZF_fun_from_total by simp
  then show II: ∀n ∈ ℤ. (m^S)(n) = m·n using ZF_fun_from_tot_val
    by simp
  { fix n k
    assume A2: n∈ℤ  k∈ℤ
    with A1 have T: m·n ∈ ℤ  m·k ∈ ℤ
      using Int_ZF_1_1_L5 by auto
    from A1 A2 II T  have δ(m^S,n,k) = m·k - m·k
      using Int_ZF_1_1_L5 Int_ZF_1_1_L1 Int_ZF_1_2_L3
      by simp
    also from T have ... = 0 using Int_ZF_1_1_L4
      by simp
    finally have δ(m^S,n,k) = 0 by simp
    then have abs(δ(m^S,n,k)) ≤ 0
      using Int_ZF_2_L18 int_zero_one_are_int int_ord_is_refl refl_def
      by simp
  } then have ∀n∈ℤ.∀k∈ℤ. abs(δ(m^S,n,k)) ≤ 0
    by simp
  with I show  m^S ∈ 𝒮 by (rule Int_ZF_2_1_L5)
qed
```

For any slope f there is an integer m such that there is some slope g that is almost equal to m^S and dominates f in the sense that $f \leq g$ on positive integers (which implies that either g is almost equal to f or $g-f$ is a positive slope. This will be used in `Real_ZF_1.thy` to show that for any real number there is an integer that (whose real embedding) is greater or equal.

```
lemma (in int1) Int_ZF_2_5_L2: assumes A1: f ∈ 𝒮
  shows ∃m∈ℤ. ∃g∈𝒮. (m^𝒮∼g ∧ (f∼g ∨ g+(-f) ∈ 𝒮₊))
proof -
  from A1 have
    ∃m k. m∈ℤ ∧ k∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ m·abs(p)+k)
    using Arthan_Lem_8 by simp
  then obtain m k where I: m∈ℤ and II: k∈ℤ and
    III: ∀p∈ℤ. abs(f(p)) ≤ m·abs(p)+k
    by auto
  let g = {⟨n,m^𝒮(n) +k⟩. n∈ℤ}
  from I have IV: m^𝒮 ∈ 𝒮 using Int_ZF_2_5_L1 by simp
  with II have V: g∈𝒮 and VI: m^𝒮∼g using Int_ZF_2_1_L33
    by auto
  { fix n assume A2: n∈ℤ₊
    with A1 have f(n) ∈ ℤ
      using Int_ZF_2_1_L2B PositiveSet_def by simp
    then have f(n) ≤ abs(f(n)) using Int_ZF_2_L19C
      by simp
    moreover
    from III A2 have abs(f(n)) ≤ m·abs(n) + k
      using PositiveSet_def by simp
    with A2 have abs(f(n)) ≤ m·n+k
      using Int_ZF_1_5_L4A by simp
    ultimately have f(n) ≤ m·n+k
      by (rule Int_order_transitive)
    moreover
    from II IV A2 have g(n) = (m^𝒮)(n)+k
      using Int_ZF_2_1_L33 PositiveSet_def by simp
    with I A2 have g(n) = m·n+k
      using Int_ZF_2_5_L1 PositiveSet_def by simp
    ultimately have f(n) ≤ g(n)
      by simp
  } then have ∀n∈ℤ₊. f(n) ≤ g(n)
    by simp
  with A1 V have f∼g ∨ g + (-f) ∈ 𝒮₊
    using Int_ZF_2_3_L4C by simp
  with I V VI show thesis by auto
qed
```

The negative of an integer embeds in slopes as a negative of the orgiginal embedding.

```
lemma (in int1) Int_ZF_2_5_L3: assumes A1:  m ∈ ℤ
  shows (-m)^𝒮 = -(m^𝒮)
proof -
  from A1 have (-m)^𝒮: ℤ→ℤ and (-(m^𝒮)): ℤ→ℤ
    using Int_ZF_1_1_L4 Int_ZF_2_5_L1 AlmostHoms_def Int_ZF_2_1_L12
    by auto
  moreover have ∀n∈ℤ. ((-m)^𝒮)(n) = (-(m^𝒮))(n)
  proof
```

```
    fix n assume A2: n∈ℤ
    with A1 have
      ((-m)^S)(n) = (-m)·n
      (-(m^S))(n) = -(m·n)
      using Int_ZF_1_1_L4 Int_ZF_2_5_L1 Int_ZF_2_1_L12A
      by auto
    with A1 A2 show ((-m)^S)(n) = (-(m^S))(n)
      using Int_ZF_1_1_L5 by simp
  qed
  ultimately show (-m)^S = -(m^S) using fun_extension_iff
    by simp
qed
```

The sum of embeddings is the embeding of the sum.

```
lemma (in int1) Int_ZF_2_5_L3A: assumes A1: m∈ℤ  k∈ℤ
  shows (m^S) + (k^S) = ((m+k)^S)
proof -
  from A1 have T1: m+k ∈ ℤ using Int_ZF_1_1_L5
    by simp
  with A1 have T2:
    (m^S) ∈ 𝒮  (k^S) ∈ 𝒮
    (m+k)^S  ∈ 𝒮
    (m^S) + (k^S) ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L12C by auto
  then have
    (m^S) + (k^S) : ℤ→ℤ
    (m+k)^S : ℤ→ℤ
    using AlmostHoms_def by auto
  moreover have ∀n∈ℤ. ((m^S) + (k^S))(n) = ((m+k)^S)(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 T1 T2 have  ((m^S) + (k^S))(n) = (m+k)·n
      using Int_ZF_2_1_L12B Int_ZF_2_5_L1 Int_ZF_1_1_L1
      by simp
    also from T1 A2 have ... = ((m+k)^S)(n)
      using Int_ZF_2_5_L1 by simp
    finally show ((m^S) + (k^S))(n) = ((m+k)^S)(n)
      by simp
  qed
  ultimately show (m^S) + (k^S) = ((m+k)^S)
    using fun_extension_iff by simp
qed
```

The composition of embeddings is the embeding of the product.

```
lemma (in int1) Int_ZF_2_5_L3B: assumes A1: m∈ℤ  k∈ℤ
  shows (m^S) ∘ (k^S) = ((m·k)^S)
proof -
  from A1 have T1: m·k ∈ ℤ using Int_ZF_1_1_L5
    by simp
```

```
  with A1 have T2:
    (m^S) ∈ 𝒮   (k^S) ∈ 𝒮
    (m·k)^S   ∈ 𝒮
    (m^S) ∘ (k^S) ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L11 by auto
  then have
    (m^S) ∘ (k^S) : ℤ→ℤ
    (m·k)^S  : ℤ→ℤ
    using AlmostHoms_def by auto
  moreover have ∀n∈ℤ. ((m^S) ∘ (k^S))(n) = ((m·k)^S)(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 T2 have
      ((m^S) ∘ (k^S))(n) = (m^S)(k·n)
       using Int_ZF_2_1_L10 Int_ZF_2_5_L1 by simp
    moreover
    from A1 A2 have k·n ∈ ℤ using Int_ZF_1_1_L5
      by simp
    with A1 A2 have (m^S)(k·n) = m·k·n
      using Int_ZF_2_5_L1 Int_ZF_1_1_L7 by simp
    ultimately have ((m^S) ∘ (k^S))(n) = m·k·n
      by simp
    also from T1 A2 have m·k·n = ((m·k)^S)(n)
      using Int_ZF_2_5_L1 by simp
    finally show ((m^S) ∘ (k^S))(n) = ((m·k)^S)(n)
      by simp
  qed
  ultimately show (m^S) ∘ (k^S) = ((m·k)^S)
    using fun_extension_iff by simp
qed
```

Embedding integers in slopes preserves order.

```
lemma (in int1) Int_ZF_2_5_L4: assumes A1:  m≤n
  shows (m^S) ∼ (n^S) ∨ (n^S)+(-(m^S)) ∈ 𝒮₊
proof -
  from A1 have m^S ∈ 𝒮 and n^S ∈ 𝒮
    using Int_ZF_2_L1A Int_ZF_2_5_L1 by auto
  moreover from A1 have ∀k∈ℤ₊. (m^S)(k) ≤ (n^S)(k)
    using Int_ZF_1_3_L13B Int_ZF_2_L1A PositiveSet_def Int_ZF_2_5_L1
    by simp
  ultimately show thesis using Int_ZF_2_3_L4C
    by simp
qed
```

We aim at showing that $m \mapsto m^S$ is an injection modulo the relation of almost equality. To do that we first show that if m^S has finite range, then $m = 0$.

```
lemma (in int1) Int_ZF_2_5_L5:
  assumes m∈ℤ and m^S ∈ FinRangeFunctions(ℤ,ℤ)
```

```
  shows m=0
  using assms FinRangeFunctions_def Int_ZF_2_5_L1 AlmostHoms_def
    func_imagedef Int_ZF_1_6_L8 by simp
```

Embeddings of two integers are almost equal only if the integers are equal.

```
lemma (in int1) Int_ZF_2_5_L6:
  assumes A1: m∈ℤ  k∈ℤ and A2: (mS) ∼ (kS)
  shows m=k
proof -
  from A1 have T: m-k ∈ ℤ using Int_ZF_1_1_L5 by simp
  from A1 have (-(kS)) =  ((-k)S)
    using Int_ZF_2_5_L3 by simp
  then have mS + (-(kS)) = (mS) + ((-k)S)
    by simp
  with A1 have mS + (-(kS)) = ((m-k)S)
    using Int_ZF_1_1_L4 Int_ZF_2_5_L3A by simp
  moreover from A1 A2 have mS + (-(kS)) ∈ FinRangeFunctions(ℤ,ℤ)
    using Int_ZF_2_5_L1 Int_ZF_2_1_L9D by simp
  ultimately have (m-k)S ∈ FinRangeFunctions(ℤ,ℤ)
    by simp
  with T have m-k = 0 using Int_ZF_2_5_L5
    by simp
  with A1 show m=k by (rule Int_ZF_1_L15)
qed
```

Embedding of 1 is the identity slope and embedding of zero is a finite range function.

```
lemma (in int1) Int_ZF_2_5_L7: shows
  1S = id(ℤ)
  0S ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  have id(ℤ) = {⟨x,x⟩. x∈ℤ}
    using id_def by blast
  then show 1S = id(ℤ) using Int_ZF_1_1_L4 by simp
  have {0S(n). n∈ℤ} = {0·n. n∈ℤ}
    using int_zero_one_are_int Int_ZF_2_5_L1 by simp
  also have ... = {0} using Int_ZF_1_1_L4 int_not_empty
    by simp
  finally have {0S(n). n∈ℤ} = {0} by simp
  then have {0S(n). n∈ℤ} ∈ Fin(ℤ)
    using int_zero_one_are_int Finite1_L16 by simp
  moreover have 0S: ℤ→ℤ
    using int_zero_one_are_int Int_ZF_2_5_L1 AlmostHoms_def
    by simp
  ultimately show 0S ∈ FinRangeFunctions(ℤ,ℤ)
    using Finite1_L19 by simp
qed
```

A somewhat technical condition for a embedding of an integer to be "less or equal" (in the sense apriopriate for slopes) than the composition of a slope and another integer (embedding).

```
lemma (in int1) Int_ZF_2_5_L8:
  assumes A1: f ∈ 𝒮 and A2: N ∈ ℤ  M ∈ ℤ and
  A3: ∀n∈ℤ₊. M·n ≤ f(N·n)
  shows M^𝒮 ∼ f∘(N^𝒮) ∨  (f∘(N^𝒮)) + (-(M^𝒮)) ∈ 𝒮₊
proof -
  from A1 A2 have M^𝒮 ∈ 𝒮  f∘(N^𝒮) ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L11 by auto
  moreover from A1 A2 A3 have ∀n∈ℤ₊. (M^𝒮)(n) ≤ (f∘(N^𝒮))(n)
    using Int_ZF_2_5_L1 PositiveSet_def Int_ZF_2_1_L10
    by simp
  ultimately show thesis using Int_ZF_2_3_L4C
    by simp
qed
```

Another technical condition for the composition of a slope and an integer (embedding) to be "less or equal" (in the sense apriopriate for slopes) than embedding of another integer.

```
lemma (in int1) Int_ZF_2_5_L9:
  assumes A1: f ∈ 𝒮 and A2: N ∈ ℤ  M ∈ ℤ and
  A3: ∀n∈ℤ₊.  f(N·n) ≤ M·n
  shows f∘(N^𝒮) ∼ (M^𝒮) ∨ (M^𝒮) + (-(f∘(N^𝒮))) ∈ 𝒮₊
proof -
  from A1 A2 have f∘(N^𝒮) ∈ 𝒮  M^𝒮 ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L11 by auto
  moreover from A1 A2 A3 have ∀n∈ℤ₊. (f∘(N^𝒮))(n) ≤ (M^𝒮)(n)
    using Int_ZF_2_5_L1 PositiveSet_def Int_ZF_2_1_L10
    by simp
  ultimately show thesis using Int_ZF_2_3_L4C
    by simp
qed

end
```

60 Integer powers of group elements

```
theory IntGroup_ZF imports Group_ZF_2 Int_ZF_1

begin
```

In the `Monoid_ZF_1` theory we consider multiplicities of $n \cdot x$ of monoid elements, i.e. special cases of expressions of the form $x_1 \oplus x_2 \oplus \ldots \oplus x_n$ where $x_i = x$ for $i = 1..n$. In the group context where we usually use multiplicative notation this translates to the "power" x^n where $n \in \mathbb{N}$, see also section "Product of a list of group elements" in the `Group_ZF` theory. In the group setting the notion of raising an element to natural power can be naturally generalized to the notion of raising an element to an integer power.

60.1 Properties of natural powers of an element and its inverse

The integer power is defined in terms of natural powers of an element and its inverse. In this section we study properties of expressions $(x^n) \cdot (x^{-1})^k$, where x is a group element and n, k are natural numbers.

The natural power of an element's inverse is the inverse of the power of the element.

```
lemma (in group0) nat_pow_inverse: assumes n∈nat x∈G
  shows pow(n,x⁻¹) = (pow(n,x))⁻¹
proof -
  from assms(1) have n∈nat and pow(0,x⁻¹) = (pow(0,x))⁻¹
    using monoid.nat_mult_zero group_inv_of_one by auto
  moreover from assms(2) have
    ∀k∈nat. pow(k,x⁻¹) = (pow(k,x))⁻¹ ⟶ pow(k #+ 1,x⁻¹) = (pow(k #+
1,x))⁻¹
    using nat_pow_add_one inverse_in_group group_inv_of_two monoid.nat_mult_type
      by simp
  ultimately show thesis by (rule ind_on_nat1)
qed
```

The natural power of x multiplied by the same power of x^{-1} cancel out to give the neutral element of the group.

```
corollary (in group0) nat_pow_inv_cancel: assumes n∈nat x∈G
  shows pow(n,x)·pow(n,x⁻¹) = 1 pow(n,x⁻¹)·pow(n,x) = 1
  using nat_pow_inverse assms group0_2_L6 monoid.nat_mult_type assms by
auto
```

If $k \leq n$ are natural numbers and x an element of the group, then $x^n \cdot (x^{-1})^k = x^(k-n)$.

```
lemma (in group0) nat_pow_cancel_less: assumes n∈nat k≤n x∈G
  shows pow(n,x)·pow(k,x⁻¹) = pow(n #- k,x)
proof -
  from assms have k∈nat n #- k ∈ nat and
    pow((n #- k),x)∈G pow(k,x)∈G pow(k,x⁻¹)∈G
    using leq_nat_is_nat diff_type monoid.nat_mult_type
      inverse_in_group by simp_all
  from assms(3) ‹k∈nat› ‹n #- k ∈ nat› have
    pow((n #- k) #+ k,x) = pow((n #- k),x)·pow(k,x)
    using monoid.nat_mult_add by simp
  with assms(1,2) ‹pow((n #- k),x)∈G› ‹pow(k,x)∈G› ‹pow(k,x⁻¹)∈G›
    have pow(n,x)·pow(k,x⁻¹) = pow(n #- k,x)·(pow(k,x)·pow(k,x⁻¹))
      using add_diff_inverse2 group_oper_assoc by simp
  with assms(3) ‹k∈nat› ‹pow((n #- k),x) ∈ G› show thesis
    using nat_pow_inv_cancel(1) group0_2_L2 by simp
qed
```

If $k \leq n$ are natural numbers and x an element of the group, then $(x^{-1})^n \cdot x^k = (x^{-1})^(k-n)$.

```
lemma (in group0) nat_pow_cancel_less1: assumes n∈nat k≤n x∈G
  shows pow(n,x⁻¹)·pow(k,x) = pow(n #- k,x⁻¹)
proof -
  from assms have pow(n,x⁻¹)·pow(k,(x⁻¹)⁻¹) = pow(n #- k,x⁻¹)
    using inverse_in_group nat_pow_cancel_less by simp
  with assms(3) show thesis using group_inv_of_inv by simp
qed
```

If $k \leq n$ are natural numbers and x an element of the group, then $x^k \cdot (x^{-1})^n = (x^{-1})^(k-n)$.

```
lemma (in group0) nat_pow_cancel_more: assumes n∈nat k≤n x∈G
  shows pow(k,x⁻¹)·pow(n,x) = pow(n #- k,x)
proof -
  from assms have k∈nat n #- k ∈ nat and
    pow((n #- k),x) ∈ G pow(k,x) ∈ G pow(k,x⁻¹) ∈ G
    using leq_nat_is_nat diff_type monoid.nat_mult_type inverse_in_group

      by simp_all
  from assms(3) ‹k∈nat› ‹n #- k ∈ nat› have
    pow(k #+ (n #- k),x) = pow(k,x)·pow((n #- k),x)
    using monoid.nat_mult_add by simp
  with assms(1,2) ‹pow((n #- k),x)∈G› ‹pow(k,x)∈G› ‹pow(k,x⁻¹)∈G›
  have pow(k,x⁻¹)·pow(n,x) = (pow(k,x⁻¹)·pow(k,x))·pow(n #- k,x)
    using add_diff_inverse group_oper_assoc by simp
  with assms(3) ‹k∈nat› ‹pow((n #- k),x) ∈ G› show thesis
    using nat_pow_inv_cancel(2) group0_2_L2 by simp
qed
```

If $k \leq n$ are natural numbers and x an element of the group, then $(x^{-1})^k \cdot x^n = x^{(k-n)}$.

```
lemma (in group0) nat_pow_cancel_more1: assumes n∈nat k≤n x∈G
  shows pow(k,x)·pow(n,x⁻¹) = pow(n #- k,x⁻¹)
proof -
  from assms have pow(k,(x⁻¹)⁻¹)·pow(n,x⁻¹) = pow(n #- k,x⁻¹)
    using inverse_in_group nat_pow_cancel_more by simp
  with assms(3) show thesis using group_inv_of_inv by simp
qed
```

The power to a product is the power of the power.

```
lemma (in group0) nat_pow_mult:
  assumes z₁∈nat z₂∈nat g∈G
  shows pow(z₁ #* z₂,g) = pow(z₂,pow(z₁,g))
proof -
  have B: pow(z₁ #* 0,g) = pow(0,pow(z₁,g)) by simp
  { fix k assume k∈nat and I: pow(z₁ #* k,g) = pow(k,pow(z₁,g))
    have z₁ #* (k #+ 1) = (z₁ #* k) #+ (z₁ #* 1)
```

```
      by (rule add_mult_distrib_left)
    then have pow(z₁ #* (k #+ 1),g) = pow((z₁ #* k) #+ (z₁ #* 1),g)
      by (rule same_constr)
    with assms(1,3) ‹k∈nat› I have pow(z₁ #* (k #+ 1),g)=pow(k #+ 1,pow(z₁,g))
      using mult_1_right mult_type nat_pow_sum_exps
        nat_pow_type nat_pow_add_one by simp
  } hence ∀k∈nat. pow(z₁ #* k,g) = pow(k,pow(z₁,g)) ⟶
           pow(z₁ #* (k #+ 1),g) = pow(k #+ 1,pow(z₁,g))
    by simp
  with assms(2) B show thesis by (rule ind_on_nat1)
qed
```

60.2 Integer powers

In this section we introduce notation basic properties of integer power in group context. The goal is to show that the power homomorphism property: if x is an element of the group and n, m are integers then $x^{n+m} = x^n \cdot x^m$.

We extend the `group0` context with some notation from `int0` context. Since we inherit the multiplicative notation from the `group0` context the integer "one" is denoted $\mathbf{1}_Z$ rather than just $\mathbf{1}$ (which is the group's neutral element).

```
locale group_int0 = group0 +

  fixes ints (ℤ)
  defines ints_def [simp]: ℤ ≡ int

  fixes ia (infixl + 69)
  defines ia_def [simp]: a+b ≡ IntegerAddition⟨ a,b⟩

  fixes iminus (- _ 72)
  defines rminus_def [simp]: -a ≡ GroupInv(ℤ,IntegerAddition)(a)

  fixes isub (infixl - 69)
  defines isub_def [simp]: a-b ≡ a+ (- b)

  fixes imul (infixl ·Z 69)
  defines imul_def [simp]: a·Zb ≡ IntegerMultiplication⟨a,b⟩

  fixes setneg (- _ 72)
  defines setneg_def [simp]: -A ≡ GroupInv(ℤ,IntegerAddition)(A)

  fixes izero (0)
  defines izero_def [simp]: 0 ≡ TheNeutralElement(ℤ,IntegerAddition)

  fixes ione (1Z)
  defines ione_def [simp]: 1Z ≡ TheNeutralElement(ℤ,IntegerMultiplication)

  fixes itwo (2)
  defines itwo_def [simp]: 2 ≡ 1Z+1Z
```

```
  fixes ithree (3)
  defines ithree_def [simp]: 3 ≡ 2+1_Z

  fixes nonnegative (ℤ⁺)
  defines nonnegative_def [simp]:
  ℤ⁺ ≡ Nonnegative(ℤ,IntegerAddition,IntegerOrder)

  fixes positive (ℤ₊)
  defines positive_def [simp]:
  ℤ₊ ≡ PositiveSet(ℤ,IntegerAddition,IntegerOrder)

  fixes abs
  defines abs_def [simp]:
  abs(m) ≡ AbsoluteValue(ℤ,IntegerAddition,IntegerOrder)(m)

  fixes lesseq (infix ≤ 60)
  defines lesseq_def [simp]: m ≤ n ≡ ⟨m,n⟩ ∈ IntegerOrder

  fixes sless (infix < 68)
  defines sless_def [simp]: a < b ≡ a≤b ∧ a≠b

  fixes interval (infix .. 70)
  defines interval_def [simp]: m..n ≡ Interval(IntegerOrder,m,n)

  fixes maxf
  defines maxf_def [simp]: maxf(f,A) ≡ Maximum(IntegerOrder,f(A))

  fixes minf
  defines minf_def [simp]: minf(f,A) ≡ Minimum(IntegerOrder,f(A))

  fixes oddext (_ °)
  defines oddext_def [simp]: f° ≡ OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
```

Next define notation for the integer power powz(z,x). The difficulty here is that in ZF set theory nonnegative integers and natural numbers are different things. So, we use the notion of zmagnitude defined in the standard Isabelle/ZF Int theory. For an integer number z, zmagnitude(z) is like absolute value of z but interpreted as a natural number. Hence, we define the integer power powz(z,x) as x raised to the magnitude of z if z is nonnegative or x^{-1} raised to the same natural power otherwise.

```
definition (in group_int0) powz where
  powz(z,x) ≡ pow(zmagnitude(z),if 0≤z then x else x⁻¹)
```

We bring in all the results about integers in int0 with the notation included in group_int0.

```
sublocale group_int0 < ints:int0 ℤ ia iminus isub imul setneg izero ione
itwo
```

```
  ithree nonnegative positive abs lesseq sless interval maxf minf oddext
by auto
```

An integer power of a group element is in the group.

```
lemma (in group_int0) powz_type: assumes z∈ℤ x∈G shows powz(z,x) ∈
G
  using assms zmagnitude_type monoid.nat_mult_type inverse_in_group
  unfolding powz_def by simp
```

A group element raised to (integer) zero'th power is equal to the group's neutral element. An element raised to (integer) power one is the same element.

```
lemma (in group_int0) int_power_zero_one: assumes x∈G
  shows powz(0,x) = 1 and powz(1_Z,x) = x
  using assms ints.Int_ZF_1_L8A(1) ints.int_ord_is_refl1 ints.zmag_zero_one

    monoid.nat_mult_zero ints.Int_ZF_2_L16A(1) monoid.nat_mult_one
    unfolding powz_def by auto
```

The neutral raised to any (integer) power is equal to the group's neutral element.

```
lemma (in group_int0) int_power_neutral: assumes z∈ℤ
  shows powz(z,1) = 1
  using monoid.nat_mult_neutral group_inv_of_one zmagnitude_type
    unfolding powz_def by auto
```

If x is an element of the group and z_1, z_2 are nonnegative integers then $x^{z_1+z_2} = x^{z_1} \cdot x^{z_2}$, i.e. the power homomorphism property holds.

```
lemma (in group_int0) powz_hom_nneg_nneg: assumes 0≤z₁ 0≤z₂ x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
  using assms ints.add_nonneg_ints(1,2) monoid.nat_mult_add
  unfolding powz_def by simp
```

If x is an element of the group and z_1, z_2 are negative integers then the power homomorphism property holds.

```
lemma (in group_int0) powz_hom_neg_neg:
  assumes z₁<0 z₂<0 x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
  using assms ints.neg_not_nonneg ints.add_neg_ints(1,2)
    inverse_in_group monoid.nat_mult_add
  unfolding powz_def by simp
```

When the integers are of different signs we further split into cases depending on which magnitude is greater. If x is an element of the group and z_1 is nonnegative, z_2 is negative and $|z_2| \leq |z_1|$ then the power homomorphism property holds. The proof of this lemma is presented with more detail than necessary, to show the schema of the proofs of the remaining lemmas that we let Isabelle prove automatically.

```
lemma (in group_int0) powz_hom_nneg_neg1:
  assumes 0≤z₁  z₂<0 zmagnitude(z₂) ≤ zmagnitude(z₁) x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
proof -
  let m₁ = zmagnitude(z₁)
  let m₂ = zmagnitude(z₂)
  from assms(1,2,3) have powz(z₁+z₂,x) = pow(zmagnitude(z₁+z₂),x)
    using ints.add_nonneg_neg1(1) unfolding powz_def by simp
  also from assms(1,2,3) have ... =  pow(m₁ #- m₂,x)
    using ints.add_nonneg_neg1(2) by simp
  also from assms(3,4) have ... = pow(m₁,x)·pow(m₂,x⁻¹)
    using nat_pow_cancel_less by simp
  also from assms(1,2) have ... = powz(z₁,x)·powz(z₂,x)
    using ints.neg_not_nonneg unfolding powz_def by simp
  finally show thesis by simp
qed
```

If x is an element of the group and z_1 is nonnegative, z_2 is negative and $|z_1| < |z_2|$ then the power homomorphism property holds.

```
lemma (in group_int0) powz_hom_nneg_neg2:
  assumes 0≤z₁  z₂<0  zmagnitude(z₁) < zmagnitude(z₂) x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
  using assms ints.add_nonneg_neg2 ints.neg_not_nonneg leI nat_pow_cancel_more1
  unfolding powz_def by simp
```

If x is an element of the group and z_1 is negative, z_2 is nonnegative and $|z_1| \leq |z_2|$ then the power homomorphism property holds.

```
lemma (in group_int0) powz_hom_neg_nneg1:
  assumes z₁<0 0≤z₂ zmagnitude(z₁) ≤ zmagnitude(z₂) x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
  using assms ints.add_neg_nonneg1 nat_pow_cancel_more ints.neg_not_nonneg
  unfolding powz_def by simp
```

If x is an element of the group and z_1 is negative, z_2 is nonnegative and $|z_2| < |z_1|$ then the power homomorphism property holds.

```
lemma (in group_int0) powz_hom_neg_nneg2:
  assumes z₁<0 0≤z₂ zmagnitude(z₂) < zmagnitude(z₁) x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
  using assms ints.add_neg_nonneg2 ints.neg_not_nonneg leI nat_pow_cancel_less1
  unfolding powz_def by simp
```

The next theorem collects the results from the above lemmas to show the power homomorphism property holds for any pair of integer numbers and any group element.

```
theorem (in group_int0) powz_hom_prop: assumes z₁∈ℤ z₂∈ℤ x∈G
  shows powz(z₁+z₂,x) = powz(z₁,x)·powz(z₂,x)
proof -
  from assms(1,2) have
```

```
  (0≤z₁ ∧ 0≤z₂) ∨ (z₁<0 ∧ z₂<0) ∨
   (0≤z₁ ∧ z₂<0 ∧ zmagnitude(z₂) ≤ zmagnitude(z₁)) ∨
   (0≤z₁ ∧ z₂<0 ∧ zmagnitude(z₁) < zmagnitude(z₂)) ∨
   (z₁<0 ∧ 0≤z₂ ∧ zmagnitude(z₁) ≤ zmagnitude(z₂)) ∨
   (z₁<0 ∧ 0≤z₂ ∧ zmagnitude(z₂) < zmagnitude(z₁))
   using ints.int_pair_6cases by simp
  with assms(3) show thesis
   using powz_hom_nneg_nneg powz_hom_neg_neg
   powz_hom_nneg_neg1 powz_hom_nneg_neg2
   powz_hom_neg_nneg1 powz_hom_neg_nneg2
   by blast
qed
```

If x is an element of the group and z_1, z_2 are integers then $x^{z_1 z_2} = (x^{z_2})^{z_2}$.

```
lemma (in group_int0) powz_mult: assumes z₁∈ℤ z₂∈ℤ x∈G
  shows powz(z₁·ℤz₂,x) = powz(z₂,powz(z₁,x))
proof -
  from assms have
   A: powz(z₁·ℤz₂,x) = pow(zmagnitude(z₂),pow(zmagnitude(z₁),if 0≤(z₁·ℤz₂)
then x else x⁻¹))
   using ints.zmagnitud_mult nat_pow_mult zmagnitude_type inverse_in_group
   unfolding powz_def by auto
  { assume C: 0≤z₁
   with assms(1,2) have L: 0≤z₂ ⟶ 0≤(z₁·ℤz₂) using ints.Int_ZF_1_3_L2
     by auto
   { assume P: ¬(0≤z₂)
     with assms(2) have Q: z₂<0
       using ints.Int_ZF_2_L19(4) ints.int_zero_one_are_int(1) by auto
     with assms(1,2) C have U: (z₁·ℤz₂)≤0
       using ints.Int_ZF_1_3_L12 ints.Int_ZF_1_1_L5(5) by force
     { assume D: (z₁·ℤz₂) = 0
       with assms(1,2) Q have z₁ = 0 using ints.int_has_no_zero_divs
         unfolding HasNoZeroDivs_def ints_def sless_def by auto
       with assms D have powz(z₁·ℤz₂,x) = powz(z₂,powz(z₁,x))
         using int_power_zero_one(1) int_power_neutral by simp
     } moreover
     { assume z₁·ℤz₂ ≠ 0
       with assms(3) A C P U have powz(z₁·ℤz₂,x) = powz(z₂,powz(z₁,x))
         unfolding powz_def using ints.neg_not_nonneg nat_pow_inverse
zmagnitude_type
         by auto
     }
     ultimately have powz(z₁·ℤz₂,x) = powz(z₂,powz(z₁,x)) by blast
   } moreover
   { assume 0≤z₂
     with A C L have powz(z₁·ℤz₂,x) = powz(z₂,powz(z₁,x))
       unfolding powz_def by auto
   } ultimately have powz(z₁·ℤz₂,x) = powz(z₂,powz(z₁,x)) by auto
  } moreover
```

```
  { assume C: ¬(0≤z₁)
    with assms(1) have Q: z₁<0
      using ints.Int_ZF_2_L19(4) ints.int_zero_one_are_int(1)
        by auto
    then have L: 0≤z₂ ⟶ (z₁·Z z₂)≤0 using ints.Int_ZF_1_3_L12
      by auto
    { assume D: (z₁·Z z₂) = 0
      with assms(1,2) Q have Z: z₂ = 0 using ints.int_has_no_zero_divs

        unfolding HasNoZeroDivs_def ints_def by auto
      with assms(3) D have powz(z₁·Z z₂,x) = 1 using int_power_zero_one(1)
by auto
      with assms(1,3) Z have powz(z₁·Z z₂,x) = powz(z₂,powz(z₁,x))
        using int_power_zero_one(1) powz_type by simp
    } moreover
    { assume D: (z₁·Z z₂) ≠ 0
      with L have L1: 0≤z₂ ⟶ ¬(0≤(z₁·Z z₂)) using ints.ls_not_leq by
auto
      from Q have ¬(0≤z₁) using ints.ls_not_leq by auto
      then have S: powz(z₁,x) = pow(zmagnitude(z₁),x⁻¹) unfolding powz_def
by auto
      { assume 0≤z₂
        with A L1 S have powz(z₁·Z z₂,x) = powz(z₂,powz(z₁,x)) unfolding
powz_def
          by auto
      } moreover
      { assume E: ¬(0≤z₂)
        with assms S have
          B: powz(z₂, powz(z₁,x)) = pow(zmagnitude(z₂), pow(zmagnitude(z₁),x))
          unfolding powz_def
          using nat_pow_inverse zmagnitude_type group_inv_of_inv
            zmagnitude_type monoid.nat_mult_type by simp
        from assms(1,2) C E have 0≤((-z₁)·Z (-z₂))
          using ints.Int_ZF_2_L19A(2) ints.Int_ZF_1_3_L2 ints.Int_ZF_1_1_L4(7)
          by simp
        with assms(1,2) A B have powz(z₁·Z z₂,x) = powz(z₂, powz(z₁,x))
          using ints.Int_ZF_1_1_L5(11) by simp
      }
      ultimately have powz(z₁·Z z₂,x) = powz(z₂, powz(z₁,x)) by auto
    }
    ultimately have powz(z₁·Z z₂,x) = powz(z₂, powz(z₁,x)) by blast
  }
  ultimately show powz(z₁·Z z₂,x) = powz(z₂, powz(z₁,x)) by auto
qed
```

A group element raised to power -1 is the inverse of that group element.

```
lemma (in group_int0) inpt_power_neg_one: assumes x∈G
  shows powz(-1Z,x) = x⁻¹
  using assms ints.neg_not_nonneg ints.neg_one_less_zero
```

```
    ints.zmag_opposite_same(2) ints.Int_ZF_1_L8A(2) ints.zmag_zero_one(2)

    inverse_in_group monoid.nat_mult_one
  unfolding powz_def by simp
```

Increasing the (integer) power by one is the same as multiplying by the group element.

```
lemma (in group_int0) int_power_add_one: assumes z∈ℤ x∈G
  shows powz(z+1_Z,x) = powz(z,x)·x
  using assms ints.Int_ZF_1_L8A(2) powz_hom_prop int_power_zero_one(2)
  by simp
```

For integer power taking a negative of the exponent is the same as taking inverse of the group element.

```
lemma (in group_int0) minus_exp_inv_base: assumes z∈ℤ x∈G
  shows powz(-z,x) = powz(z,x⁻¹)
proof -
  from assms(1) have 0<z ∨ z=0 ∨ z<0
    using ints.int_neg_zero_pos by simp
  moreover from assms(1) have 0<z ⟶ powz(-z,x) = powz(z,x⁻¹)
    using ints.neg_pos_int_neg ints.neg_not_nonneg ints.zmag_opposite_same(2)

    unfolding powz_def by simp
  moreover from assms(2) have z=0 ⟶ powz(-z,x) = powz(z,x⁻¹)
    using ints.Int_ZF_1_L11 int_power_zero_one(1) inverse_in_group by
simp
  moreover from assms have z<0 ⟶ powz(-z,x) = powz(z,x⁻¹)
    using ints.neg_not_nonneg ints.neg_neg_int_pos group_inv_of_inv
    ints.zmag_opposite_same(2) unfolding powz_def by simp
  ultimately show powz(-z,x) = powz(z,x⁻¹) by auto
qed
```

Integer power of a group element is the same as the inverse of the element raised to negative of the exponent.

```
lemma (in group_int0) minus_exp_inv_base1: assumes z∈ℤ x∈G
  shows powz(z,x) = powz(-z,x⁻¹)
proof -
  from assms(1) have (-z)∈ℤ using ints.int_neg_type by simp
  with assms show thesis using minus_exp_inv_base ints.neg_neg_noop
    by force
qed
```

The next context is like `group_int0` but adds the assumption that the group operation is commutative (i.e. the group is abelian).

```
locale abgroup_int0 = group_int0 +
  assumes isAbelian: P {is commutative on} G
```

In abelian groups taking a nonnegative integer power commutes with the group operation. Unfortunately we have to drop to raw set theory notation in the proof to be able to use `int0.Induction_on_int` from the `abgroup_int0` context.

```
lemma (in abgroup_int0) powz_groupop_commute0: assumes 0≤k x∈G y∈G
  shows powz(k,x·y) = powz(k,x)·powz(k,y)
proof -
  let A_Z = IntegerAddition
  let M_Z = IntegerMultiplication
  let O_Z = IntegerOrder
  from assms have ⟨0,k⟩ ∈ O_Z and powz(0,x·y) = powz(0,x)·powz(0,y)
    using group_op_closed int_power_zero_one(1) group0_2_L2 by simp_all
  moreover
  { fix m assume 0≤m and I: powz(m,x·y) = powz(m,x)·powz(m,y)
    from assms(2,3) ‹0≤m› have  m∈ℤ and x·y ∈ G
      using ints.Int_ZF_2_L1A(3) group_op_closed by simp_all
    with isAbelian assms(2,3) I have powz(m+1_Z,x·y) = powz(m+1_Z,x)·powz(m+1_Z,y)
      using int_power_add_one group0_4_L8(3) powz_type by simp
  } hence ∀m. 0≤m ∧ powz(m,x·y) = powz(m,x)·powz(m,y) ⟶
     powz(m+1_Z,x·y) = powz(m+1_Z,x)·powz(m+1_Z,y) by simp
  hence ∀m. ⟨0, m⟩ ∈ O_Z ∧ powz(m,x·y) = powz(m,x)·powz(m,y) ⟶
    powz(A_Z(⟨m,TheNeutralElement(int,M_Z)⟩),x·y) =
    powz(A_Z(⟨m,TheNeutralElement(int,M_Z)⟩),x)·powz(A_Z(⟨m,TheNeutralElement(int,M_Z)⟩),y)
    by simp
  ultimately show powz(k,x·y) = powz(k,x)·powz(k,y) by (rule int0.Induction_on_int)
qed
```

In abelian groups taking a nonpositive integer power commutes with the group operation. We could use backwards induction in the proof but it is shorter to use the nonnegative case from `powz_groupop_commute0`.

```
lemma (in abgroup_int0) powz_groupop_commute1: assumes k≤0 x∈G y∈G
  shows powz(k,x·y) = powz(k,x)·powz(k,y)
proof -
  from isAbelian assms have powz(k,x·y) = powz(-k,x⁻¹·y⁻¹)
    using ints.Int_ZF_2_L1A(2) group_op_closed minus_exp_inv_base1 group_inv_of_two
    unfolding IsCommutative_def by simp
  with assms show thesis
    using ints.Int_ZF_2_L10A inverse_in_group powz_groupop_commute0
      ints.Int_ZF_2_L1A(2) minus_exp_inv_base1 by simp
qed
```

In abelian groups taking an integer power commutes with the group operation.

```
theorem (in abgroup_int0) powz_groupop_commute: assumes z∈ℤ x∈G y∈G
  shows powz(z,x·y) = powz(z,x)·powz(z,y)
proof -
  from assms(1) have 0≤z ∨ z≤0 using ints.int_nonneg_or_nonpos
    by auto
  with assms(2,3) show thesis using powz_groupop_commute0 powz_groupop_commute1
    by blast
```

```
qed
```

For any integer n the mapping $x \mapsto x^n$ maps G into G and is a homomorphism hence an endomorphism of (G, P).

```
theorem (in abgroup_int0) powz_end: assumes n∈ℤ
  defines h ≡ {⟨x,powz(n,x)⟩. x∈G}
  shows h:G→G  Homomor(h,G,P,G,P)  h ∈ End(G,P)
proof -
  from assms show h:G→G using powz_type ZF_fun_from_total by simp
  with assms have IsMorphism(G,P,P,h)
    using ZF_fun_from_tot_val(1) group_op_closed powz_groupop_commute
    unfolding IsMorphism_def by simp
  with ‹h:G→G› show Homomor(h,G,P,G,P) and h ∈ End(G,P)
    unfolding Homomor_def End_def by simp_all
qed
```

The mapping $\mathbb{Z} \ni n \mapsto (x \mapsto x^n \in G)$ maps integers to endomorphisms of (G, P).

```
theorem (in abgroup_int0) powz_maps_int_End:
  shows {⟨n,{⟨x,powz(n,x)⟩. x∈G}⟩. n∈ℤ} : ℤ→End(G,P)
  using powz_end(3) ZF_fun_from_total by simp

end


theory IntModule_ZF imports Module_ZF Int_ZF_1 IntGroup_ZF

begin
```

61 $\mathbb{Z}$ modules

In this section we show that the integers, as a ring, have only one module structure on each abelian group. We will show that the module structure is unique, but we will also show which action is the one that defines that module structure.

When $\mathbb{Z}$ acts on a group, that action is unique.

```
lemma (in int0) action_unique:
  assumes IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,f,S₁)
and
    IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,f,S₂)
  shows S₁ = S₂
proof -
  let A_Z = IntegerAddition
  let M_Z = IntegerMultiplication
  from assms have mod0: module0(ℤ,A_Z,M_Z,G,f,S₁) module0(ℤ,A_Z,M_Z,G,f,S₂)
      unfolding module0_def IsLeftModule_def ring0_def module0_axioms_def
by auto
```

```
  { fix r assume rg: r∈ℤ
    from mod0(1) have eq: ∀g∈G. (S₁(0))(g) = TheNeutralElement(G, f)
      using module0.mult_zero by simp
    with mod0(1) have S₁(0) = {⟨g,TheNeutralElement(G,f)⟩. g∈G}
      using Int_ZF_1_L8A(1) module0.H_val_type(2) fun_is_set_of_pairs
      by force
    then have s1: S₁(0) = ConstantFunction(G,TheNeutralElement(G, f))
      unfolding ConstantFunction_def by auto
    from mod0(2) have eq: ∀g∈G. (S₂(0))(g) = TheNeutralElement(G, f)
      using module0.mult_zero by auto
    with mod0(2) have (S₂(0)):G→G using Int_ZF_1_L8A(1) module0.H_val_type
      by auto
    with eq s1 have S₁(0) = S₂(0)
      unfolding ConstantFunction_def using fun_is_set_of_pairs by force
    { assume r≤0
      then have r∈ℤ using OrderedGroup_ZF_1_L4(1) by simp
      have S₁(r) = S₂(r)
      proof -
        { fix n assume n≤0 S₁(n) = S₂(n)
          from ‹n≤0› have n∈ℤ using OrderedGroup_ZF_1_L4(1) by simp
          have (-1) ∈ ℤ using group0.inverse_in_group Int_ZF_1_T2(3)
              ring0.Ring_ZF_1_L2(2) Int_ZF_1_1_L2(3) by simp
          { fix t assume t: t∈G
            with mod0(1) ‹S₁(n) = S₂(n)› ‹n∈ℤ› ‹(-1)∈ℤ› have
              S₁(n+(- 1))(t) = (f⟨(S₂(n))(t),(S₁(-1))(t)⟩)
              using  module0.module_ax2 t by simp
            moreover from mod0(1) have ex: ∀g∈G. S₁(-1)(g) = GroupInv(G,
f)(g)
              using module0.inv_module apply_type module0.H_val_type(2)
ring0.Ring_ZF_1_L2(2)
                Int_ZF_1_1_L2(3) module0.module_ax4 by simp
            moreover from mod0(1) have S₁(- 1):G→G
              using module0.H_val_type(2) ring0.Ring_ZF_1_L2(2) Int_ZF_1_1_L2(3)
                ring0.Ring_ZF_1_L3(1) Int_ZF_1_1_L2(3) by auto
            moreover
            from assms(1) have GroupInv(G, f): G→G
              using group0_2_T2 unfolding IsLeftModule_def by blast
            with ex ‹S₁(- 1):G→G› have S₁(- 1) =  GroupInv(G, f)
              using func_eq by simp
            ultimately have A: S₁(n+(-1))(t) = f⟨(S₂(n))(t),GroupInv(G,
f) t⟩
              by simp
            from assms(2) ‹(-1) ∈ ℤ› ‹n∈ℤ› mod0(2) t ‹S₁(n) = S₂(n)›
have
              (S₂(n+-1))(t) = f⟨(S₂(n))(t),(S₂(-1))(t)⟩
              using module0.module_ax2 by simp
            moreover from mod0(2) have ex: ∀g∈G. (S₂(-1))(g) = GroupInv(G,
f)(g)
              using module0.inv_module apply_type module0.H_val_type(2)
```

```
            ring0.Ring_ZF_1_L2(2) Int_ZF_1_1_L2(3) module0.module_ax4
by simp
          moreover
          from mod0(2) have S₂(- 1): G→G
            using module0.H_val_type(2) ring0.Ring_ZF_1_L2(2) Int_ZF_1_1_L2(3)
              ring0.Ring_ZF_1_L3(1) Int_ZF_1_1_L2(3) by auto
          with ex ‹GroupInv(G, f): G→G› have S₂(-1) = GroupInv(G, f)

            using func_eq by blast
          ultimately have S₂(n+(-1))(t) = f⟨(S₂(n))(t),GroupInv(G, f)(t)⟩
            by simp
          with A have (S₁(n+(- 1)))(t) = S₂(n+(-1))(t) by simp
        } hence ∀t∈G. S₁(n + - 1)(t) = S₂(n+(-1))(t) by simp
        moreover from mod0 ‹(-1) ∈ ℤ› ‹n∈ℤ› have
          S₁(n+(-1)):G→G and S₂(n+(-1)):G→G
          using group0.group_op_closed Int_ZF_1_T2(3) module0.H_val_type(2)

          by simp_all
        ultimately have S₁(n+(-1)) = S₂(n+(-1))
          using func_eq by simp
      } hence ∀n. (n≤0) ∧ (S₁(n) = S₂(n))⟶S₁(n-1) = S₂(n-1)
        by auto
      with ‹r≤0› ‹S₁(0) = S₂(0)› show S₁(r) = S₂(r)
        using Back_induct_on_int by simp
    qed
  }
  moreover
  { assume 0≤r
    then have r∈ℤ using OrderedGroup_ZF_1_L4(2) by simp
    have S₁(r) = S₂(r)
    proof -
      { fix m assume 0≤m S₁(m) = S₂(m)
        from ‹0≤m› have m∈ℤ using OrderedGroup_ZF_1_L4(2) by simp
        have 1∈ℤ using Int_ZF_1_L8A(2) by simp
        { fix t assume  t∈G
          with assms(1) ‹m∈ℤ› ‹1∈ℤ› mod0(1) ‹S₁(m) = S₂(m)› have
            S₁(m+1)t = f⟨(S₂(m))(t),t⟩
            using module0.module_ax2 module0.module_ax4 by simp
          moreover from mod0(2) ‹m∈ℤ› ‹1∈ℤ› ‹t∈G› have
            (S₂(m+1))(t) = f⟨(S₂(m))(t),t⟩
            using module0.module_ax2 module0.module_ax4 by simp
          ultimately have (S₁(m+1))(t) = (S₂(m+1))(t)
            by simp
        } hence ∀t∈G. (S₁(m+1))(t) = (S₂(m+1))(t) by simp
        moreover from mod0 ‹1∈ℤ› ‹m∈ℤ› have
          S₁(m+1):G→G and S₂(m+1):G→G
          using group0.group_op_closed Int_ZF_1_T2(3) module0.H_val_type(2)
            module0.H_val_type(2) by simp_all
```

```
        ultimately have S₁(m+1) = S₂(m+1)
          using func_eq by simp
      } hence ∀m. (0≤m) ∧ S₁(m) = S₂(m) ⟶ S₁(m+1) = S₂(m+1)
          by simp
        with ‹0≤r› ‹S₁(0) = S₂(0)› show S₁(r) = S₂(r) using Induction_on_int

          by simp
    qed
  }
  moreover have IntegerOrder {is total on} ℤ using Int_ZF_2_T1(2)
by simp
  ultimately have S₁(r) = S₂(r) unfolding IsTotal_def
    using rg int0.Int_ZF_1_L8(1) ring0.Ring_ZF_1_L2(1) Int_ZF_1_1_L2(3)

    by auto
} hence ∀r∈ℤ. S₁(r) = S₂(r) by auto
moreover from assms have S₁:ℤ→End(G,f) S₂:ℤ→End(G,f)
  using module_action_type(3) by simp_all
ultimately show S₁ = S₂ using func_eq by simp
qed

sublocale abelian_group < int0_abgroup: abgroup_int0 G P 1 groper inv
  listprod pow int λx y. IntegerAddition⟨x,y⟩ λx. GroupInv(int,IntegerAddition)x
  λx y. IntegerAddition⟨x,GroupInv(int,IntegerAddition)y⟩
  λx y. IntegerMultiplication⟨x,y⟩ λA. GroupInv(int,IntegerAddition)(A)
  $# 0 $# 1 $# 1 $+ $# 1 ($# 1 $+ $# 1) $+ $# 1 Nonnegative(int,IntegerAddition,IntegerOrde

  PositiveSet(int,IntegerAddition,IntegerOrder) λm. AbsoluteValue(int,IntegerAddition,Integ

  λx y. ⟨x,y⟩∈IntegerOrder λx y. ⟨x,y⟩∈IntegerOrder ∧ x≠y λx y. Interval(IntegerOrder,x,y)
  λf A. Maximum(IntegerOrder,f(A)) λf A. Minimum(IntegerOrder,f(A))
  λf. OddExtension(int,IntegerAddition,IntegerOrder,f)
  apply auto
  unfolding abgroup_int0_def group_int0_def abgroup_int0_axioms_def
  apply auto using group0_axioms apply simp using abelian_group_axioms
unfolding abelian_group_def abelian_group_axioms_def apply simp
  using int0.Int_ZF_1_L8 apply simp using int0.Int_ZF_1_L8 apply simp
  using int0.Int_ZF_1_L2(1) apply simp
  using int0.Int_ZF_1_L2(1) apply simp done

definition (in abelian_group)
  powz(z,n) ≡ int0_abgroup.powz(z,n)
```

The action is an group homomorphism between $(\mathbb{Z}, +)$ and (G, P)

```
lemma(in abelian_group) group_action_int_add_morphism:
  defines S ≡ {⟨z,{⟨x,powz(z,x)⟩. x∈G}⟩. z∈int}
  shows ∀r∈int. ∀s∈int. ∀g∈G. S  (IntegerAddition ⟨r, s⟩)  g = P  ⟨(S
r)  g, (S  s)  g⟩
proof(safe)
```

```
  have S_fun:S:int→End(G,P) using int0_abgroup.powz_maps_int_End
    unfolding S_def powz_def by blast
  fix r s g assume as:r:int s:int g:G
  from as(1) int_cases obtain n where n:n∈nat r=$#n ∨ r=$- $# succ(n)
by blast
  from as(2) int_cases obtain m where m:m∈nat s=$#m ∨ s=$- $# succ(m)
by blast
  from as(1,2) have ⟨r,s⟩∈int×int unfolding Sigma_def by blast
  then have int:IntegerAddition⟨r, s⟩∈int
    using apply_type[of IntegerAddition int×int λ_. int ⟨r,s⟩]
    by (simp only: Int_ZF_1_L1(1))
  then have ⟨IntegerAddition⟨r,s⟩,{⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩. x∈G}⟩∈S
    unfolding S_def by blast
  then have Q:S(IntegerAddition⟨r,s⟩) = {⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩.
x∈G}
    using apply_equality[OF _ S_fun] by blast
  have SS_fun:S(IntegerAddition⟨r,s⟩):G→G
    using apply_type[OF S_fun int] unfolding End_def by (simp only:)
  have SS_r_fun:Sr:G→G
    using apply_type[OF S_fun as(1)] unfolding End_def by (simp only:)
  have SS_s_fun:Ss:G→G
    using apply_type[OF S_fun as(2)] unfolding End_def by (simp only:)
  {
    fix x assume x∈G
    then have powz(IntegerAddition⟨r,s⟩,x) = powz(r,x)·powz(s,x)
      using int0_abgroup.powz_hom_prop[of r s x] as(1,2) unfolding powz_def
by blast
  }
  then have R:⋀y. y∈G ⟹ powz(IntegerAddition⟨r,s⟩,y) = powz(r,y)·powz(s,y)
by blast
  {
    fix t assume as:t∈{⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩. x∈G}
    then obtain tx where t:t=⟨tx,powz(IntegerAddition⟨r,s⟩,tx)⟩ tx∈G
      by blast
    from t(1) have t=⟨tx,powz(r,tx)·powz(s,tx)⟩ by (simp only:t(2) R[of
tx])
    then have t∈{⟨x,powz(r,x)·powz(s,x)⟩. x∈G} using t(2) by blast
  }
  then have A1:{⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩. x∈G} ⊆ {⟨x,powz(r,x)·powz(s,x)⟩.
x∈G} by blast
  {
    fix t assume as:t∈{⟨x,powz(r,x)·powz(s,x)⟩. x∈G}
    then obtain tx where t:t=⟨tx,powz(r,tx)·powz(s,tx)⟩ tx∈G
      by blast
    from t(1) have t=⟨tx,powz(IntegerAddition⟨r,s⟩,tx)⟩ by (simp only:t(2)
R[of tx])
    then have t∈{⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩. x∈G} using t(2) by
blast
  }
```

```
  then have A2:{⟨x,powz(r,x)·powz(s,x)⟩. x∈G} ⊆ {⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩.
x∈G}
    by blast
  from A1 A2 have A:{⟨x,powz(IntegerAddition⟨r,s⟩,x)⟩. x∈G} = {⟨x,powz(r,x)·powz(s,x)⟩.
x∈G}
    by blast
  have E:S(IntegerAddition⟨r,s⟩) = {⟨x,powz(r,x)·powz(s,x)⟩. x∈G}
    using Q by (simp only: A)
  have ⟨g,powz(r,g)·powz(s,g)⟩∈{⟨x,powz(r,x)·powz(s,x)⟩. x∈G}
    using as(3) by blast
  then have ⟨g,powz(r,g)·powz(s,g)⟩∈ S(IntegerAddition⟨r,s⟩)
    by (simp only: E)
  from apply_equality[OF this SS_fun] have B:S(IntegerAddition⟨r,s⟩)g
= powz(r,g)·powz(s,g)
    by blast
  have ⟨r,{⟨x,powz(r,x)⟩. x∈G}⟩∈S
    unfolding S_def using as(1) by blast
  then have Qr:Sr = {⟨x,powz(r,x)⟩. x∈G}
    using apply_equality[OF _ S_fun] by blast
  have ⟨g,powz(r,g)⟩∈{⟨x,powz(r,x)⟩. x∈G} using as(3) by blast
  then have ⟨g,powz(r,g)⟩∈ Sr
    by (simp only: Qr)
  from apply_equality[OF this SS_r_fun]
    have C:Srg = powz(r,g).
  have ⟨s,{⟨x,powz(s,x)⟩. x∈G}⟩∈S
    unfolding S_def using as(2) by blast
  then have Qr:Ss = {⟨x,powz(s,x)⟩. x∈G}
    using apply_equality[OF _ S_fun] by blast
  have ⟨g,powz(s,g)⟩∈{⟨x,powz(s,x)⟩. x∈G} using as(3) by blast
  then have ⟨g,powz(s,g)⟩∈ Ss
    by (simp only: Qr)
  from apply_equality[OF this SS_s_fun]
    have D:Ssg = powz(s,g).
  from B show S  (IntegerAddition ⟨r, s⟩)  g = P  ⟨(S  r)  g, (S  s)
g⟩
    unfolding groper_def by (simp only:C D)
qed
```

Same as before, but not pointwise

```
lemma(in abelian_group) group_action_int_add_morphism_fun:
  defines S ≡ {⟨z,{⟨x,powz(z,x)⟩. x∈G}⟩. z∈int}
  shows ∀r∈int. ∀s∈int. S  (IntegerAddition ⟨r, s⟩) = EndAdd(G,P)  ⟨(S
r), (S  s)⟩
proof(safe)
  have S_fun:S:int→End(G,P) using int0_abgroup.powz_maps_int_End
    unfolding S_def powz_def by blast
  fix r s assume as:r:int s:int
  {
    fix h assume h:h∈G
```

```
    then have h·1 = h using group0_2_L2 by auto
    with h have h∈P(G×G) using imageI[of ⟨h,1⟩ h P G×G]
      using apply_Pair[OF group_oper_fun, of ⟨h,1⟩]
      group0_2_L2(1) by auto
  }
  with range_image_domain[OF group_oper_fun] have G⊆ range(P) by auto
moreover
  have range(P) ⊆ G using func1_1_L5B[OF group_oper_fun]. ultimately
  have eq:range(P) = G by auto
  from S_fun have endo:Sr∈End(G,P) Ss:End(G,P)
    using apply_type[of S int λ_. End(G,P) r]
    apply (simp only:as) using apply_type[of S int λ_. End(G,P) s] S_fun
    by (simp only:as)
  with eq have SRSS:Sr:G→range(P) Ss:G→range(P) unfolding End_def by
auto
  {
    fix g assume g:g∈G
    with as group_action_int_add_morphism have EE:S(IntegerAddition ⟨r,
s⟩)g = P  ⟨S  r  g, S  s  g⟩
      unfolding S_def by blast
    with func_ZF_1_L4[OF group_oper_fun _ _ _ g, of _ Sr Ss]
     SRSS have (P {lifted to function space over} G)  ⟨S  r, S  s⟩  g
= P  ⟨S  r  g, S  s  g⟩
      by blast
    then have (P {lifted to function space over} G)  ⟨S  r, S  s⟩  g =
S(IntegerAddition ⟨r, s⟩)g
      by (simp only:EE)
  }
  then have R:⋀x. x ∈ G ⟹ (P {lifted to function space over} G)  ⟨S
r, S  s⟩  x = S(IntegerAddition ⟨r, s⟩)  x by blast
  from as have ⟨r,s⟩∈int×int unfolding Sigma_def by blast
  then have int:IntegerAddition⟨r, s⟩∈int
    using apply_type[of IntegerAddition int×int λ_. int ⟨r,s⟩]
    by (simp only: Int_ZF_1_L1(1))
  then have f:S(IntegerAddition ⟨r, s⟩):G→G using S_fun
    apply_type[of S int λ_. End(G,P) IntegerAddition  ⟨r, s⟩] unfolding
End_def S_def
    by blast
  from func_ZF_1_L3[OF group_oper_fun, of _ G] SRSS
  have (P {lifted to function space over} G)  ⟨S  r, S  s⟩ : G→range(P)
    using apply_type by auto
  from fun_extension[OF this f R] have A:S  (IntegerAddition  ⟨r, s⟩)
= (P {lifted to function space over} G)  ⟨S  r, S  s⟩
    by (simp only:)
  have ⟨Sr,Ss⟩∈End(G,P)×End(G,P) using endo by blast
  then have (P {lifted to function space over} G)  ⟨S  r, S  s⟩ = restrict(P
{lifted to function space over} G, End(G,P)×End(G,P))  ⟨S  r, S  s⟩
    using restrict[of P {lifted to function space over} G End(G,P)×End(G,P)
⟨Sr,Ss⟩] apply (simp only:) by simp
```

```
  with A show  S  (IntegerAddition  ⟨r, s⟩) = EndAdd(G, P)  ⟨S  r, S
s⟩  unfolding EndAdd_def by (simp only:)
qed
```

The action is a homomorphism between $(\mathbb{Z}, \cdot)$ and $(G \to G, \circ)$

```
lemma(in abelian_group) group_action_int_mult_morphism:
  defines S ≡ {⟨z,{⟨x,powz(z,x)⟩. x∈G}⟩. z∈int}
  shows ∀r∈int. ∀s∈int. S  (IntegerMultiplication ⟨r, s⟩) = EndMult(G,P)⟨Sr,Ss⟩
proof(safe)
  have S_fun:S:int→End(G,P) using int0_abgroup.powz_maps_int_End
    unfolding S_def powz_def by blast
  fix r s assume as:r:int s:int
  from as have int:IntegerMultiplication⟨r,s⟩∈int using
    int0_abgroup.ints.Int_ZF_1_1_L5(3) by (simp only:)
  from int have Srs:S(IntegerMultiplication⟨r,s⟩):G→G
    using apply_type[OF S_fun] unfolding End_def by blast
  from as(1) have Sr:Sr:G→G
    using apply_type[OF S_fun] unfolding End_def by blast
  from as(2) have Ss:Ss:G→G
    using apply_type[OF S_fun] unfolding End_def by blast
  {
    fix g assume g:g∈G
    from int have ⟨IntegerMultiplication⟨r,s⟩,{⟨x,powz(IntegerMultiplication⟨r,s⟩,x)⟩.
x∈G}⟩∈S
      unfolding S_def by blast
    then have S(IntegerMultiplication⟨r,s⟩) = {⟨x,powz(IntegerMultiplication⟨r,s⟩,x)⟩.
x∈G}
      using apply_equality[OF _ S_fun] by (simp only:)
    moreover

    have ⟨g,powz(IntegerMultiplication⟨r,s⟩,g)⟩∈{⟨x,powz(IntegerMultiplication⟨r,s⟩,x)⟩.
x∈G}
      using g by blast
    ultimately have ⟨g,powz(IntegerMultiplication⟨r,s⟩,g)⟩∈S(IntegerMultiplication⟨r,s⟩)
      by (simp only:)
    then have S(IntegerMultiplication⟨r,s⟩)g = powz(IntegerMultiplication⟨r,s⟩,g)
      using apply_equality[OF _ Srs] by (simp only:)
    then have S(IntegerMultiplication⟨r,s⟩)g = powz(IntegerMultiplication⟨s,r⟩,g)
      using int0.Int_ZF_1_1_L5(5) as by (simp only:)
    then have A:S(IntegerMultiplication⟨r,s⟩)g = powz(r,powz(s,g))
      using int0_abgroup.powz_mult[OF as(2,1) g] unfolding powz_def by
(simp only:)
    have ⟨s,{⟨x,powz(s,x)⟩. x∈G}⟩∈S using as(2) unfolding S_def by blast
    then have Ss = {⟨x,powz(s,x)⟩. x∈G} using apply_equality[OF _ S_fun]
      by (simp only:)
    then have ⟨g,powz(s,g)⟩∈Ss using g by auto
    then have B:(Ss)g = powz(s,g) using apply_equality Ss by auto
    have ⟨r,{⟨x,powz(r,x)⟩. x∈G}⟩∈S using as(1) unfolding S_def by blast
    then have Sr = {⟨x,powz(r,x)⟩. x∈G} using apply_equality[OF _ S_fun]
```

```
      by (simp only:)
    then have ⟨powz(s,g),powz(r,powz(s,g))⟩∈Sr using int0_abgroup.powz_type[OF
as(2) g]
      unfolding powz_def by auto
    then have C:(Sr)powz(s,g) = powz(r,powz(s,g)) using apply_equality
Sr by auto
    with A B have S(IntegerMultiplication⟨r,s⟩)g = (Sr)((Ss)g)
      by (simp only:)
    then have G:S(IntegerMultiplication⟨r,s⟩)g = ((Sr)O(Ss))g
      using comp_fun_apply[OF Ss g] by (simp only:)
    then have S(IntegerMultiplication⟨r,s⟩)g = (Composition(G)⟨Sr,Ss⟩)g
      using func_ZF_5_L2 Ss Sr by (simp only:)
  }
  then have R:⋀g. g∈G ⟹ S(IntegerMultiplication ⟨r, s⟩)g = (Composition(G)⟨Sr,Ss⟩)g
by blast
  have B:(Composition(G)⟨Sr,Ss⟩):G→G using apply_type[OF func_ZF_5_L1]
Sr Ss by auto
  have L:S  (IntegerMultiplication  ⟨r, s⟩) = Composition(G)  ⟨S  r, S
s⟩
    using fun_extension[OF Srs B R] by blast
  have ⟨Sr,Ss⟩∈End(G,P)×End(G,P) using apply_type[OF S_fun as(1)] apply_type[OF
S_fun as(2)]
    by (simp only:)
  then have restrict(Composition(G),End(G,P)×End(G,P))⟨Sr,Ss⟩ = Composition(G)
⟨S  r, S  s⟩
    using restrict[of Composition(G) End(G,P)×End(G,P) ⟨Sr,Ss⟩] by auto
  with L show S  (IntegerMultiplication  ⟨r, s⟩) = EndMult(G, P)  ⟨S
r, S  s⟩ unfolding EndMult_def S_def
    by (simp only:)
qed
```

The unit is the identity

```
lemma (in abelian_group) group_action_int_unit:
  defines S ≡ {⟨z,{⟨x,powz(z,x)⟩. x∈G}⟩. z∈int}
  shows S  TheNeutralElement(int, IntegerMultiplication) =
    TheNeutralElement(End(G, P), EndMult(G, P))
proof-
  have S_fun:S ∈ int → End(G, P) using int0_abgroup.powz_maps_int_End
    unfolding S_def powz_def by blast
  have TheNeutralElement(int, IntegerMultiplication)∈int
    using int0.int_zero_one_are_int(2).
  then have ⟨TheNeutralElement(int, IntegerMultiplication),{⟨x,powz(TheNeutralElement(int,
IntegerMultiplication),x)⟩. x∈G}⟩∈S
    unfolding S_def by blast
  then have STheNeutralElement(int, IntegerMultiplication) = {⟨x,powz(TheNeutralElement(int
IntegerMultiplication),x)⟩. x∈G}
    using apply_equality[OF _ S_fun] by blast
  then have STheNeutralElement(int, IntegerMultiplication) = {⟨x,int0_abgroup.powz($#1,x)⟩.
x∈G}
```

```
    using int0.Int_ZF_1_L8(2) unfolding powz_def by (simp only:)
  moreover
  {
    fix t assume t∈{⟨x,int0_abgroup.powz($#1,x)⟩. x∈G}
    then obtain tx where t=⟨tx,int0_abgroup.powz($#1,tx)⟩ tx∈G by blast
    then have t=⟨tx,tx⟩ using int0_abgroup.int_power_zero_one(2)[of tx]
by (simp only:)
    then have t∈id(G) using idI `tx∈G` by auto
  }
  then have K:{⟨x,int0_abgroup.powz($#1,x)⟩. x∈G} ⊆ id(G) by blast
  {
    fix t assume t∈id(G)
    {
      fix p assume p∈G t=⟨p,p⟩
      then have t=⟨p,int0_abgroup.powz($#1,p)⟩ using int0_abgroup.int_power_zero_one(2)[of
p]
        by (simp only:)
      then have t∈{⟨x,int0_abgroup.powz($#1,x)⟩. x∈G} using `p∈G` by
blast
    }
    from idE[OF `t∈id(G)` this] have t∈{⟨x,int0_abgroup.powz($#1,x)⟩.
x∈G} by (simp only:)
  }
  then have id(G) ⊆ {⟨x,int0_abgroup.powz($#1,x)⟩. x∈G}
    by blast
  with K have {⟨x,int0_abgroup.powz($#1,x)⟩. x∈G} = id(G) by blast
  ultimately have STheNeutralElement(int, IntegerMultiplication) = id(G)
by (simp only:)
  then show thesis using end_comp_monoid(2) unfolding EndMult_def by
(simp only:)
qed
```

The action defines a module

```
theorem(in abelian_group) group_action_int:
  defines S ≡ {⟨z,{⟨x,powz(z,x)⟩. x∈G}⟩. z∈int}
  shows IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,P,S)
unfolding IsLeftModule_def IsAction_def ringHomomor_def IsMorphism_def
proof(safe)
  show IsAgroup(G,P) using groupAssum.
  show P{is commutative on}G using abelian_group_axioms unfolding abelian_group_def
abelian_group_axioms_def by auto
  show S ∈ int → End(G, P) using int0_abgroup.powz_maps_int_End
    unfolding S_def powz_def by blast
  show IsAring(int, IntegerAddition, IntegerMultiplication) using int0.Int_ZF_1_1_L2(1).
  show ⋀r s. r ∈ int ⟹
          s ∈ int ⟹ S  (IntegerAddition  ⟨r, s⟩) = EndAdd(G, P)  ⟨S
r,S  s⟩
    using group_action_int_add_morphism_fun unfolding S_def by blast
  show ⋀r s. r ∈ int ⟹ s ∈ int ⟹ S  (IntegerMultiplication  ⟨r,
```

```
s⟩) =  EndMult(G, P)  ⟨S  r, S  s⟩
    using group_action_int_mult_morphism unfolding S_def by blast
  show S  TheNeutralElement(int, IntegerMultiplication) =
    TheNeutralElement(End(G, P), EndMult(G, P)) using group_action_int_unit
unfolding S_def EndMult_def.
qed
```

If there is a $\mathbb{Z}$-module on an abelian group, it is the one found in the previous result

```
corollary(in abelian_group) group_action_int_rev:
  assumes IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,P,S)
  shows S={⟨z,{⟨x,powz(z,x)⟩. x∈G}⟩. z∈int}
  using group_action_int assms int0.action_unique[of G P S {⟨z,{⟨x,powz(z,x)⟩.
x∈G}⟩. z∈int}]
  by blast

end
```

62 Construction real numbers - the generic part

```
theory Real_ZF imports Int_ZF_IML Ring_ZF_1

begin
```

The goal of the `Real_ZF` series of theory files is to provide a contruction of the set of real numbers. There are several ways to construct real numbers. Most common start from the rational numbers and use Dedekind cuts or Cauchy sequences. `Real_ZF_x.thy` series formalizes an alternative approach that constructs real numbers directly from the group of integers. Our formalization is mostly based on [2]. Different variants of this contruction are also described in [1] and [3]. I recommend to read these papers, but for the impatient here is a short description: we take a set of maps $s : Z \to Z$ such that the set $\{s(m+n) - s(m) - s(n)\}_{n,m \in Z}$ is finite (Z means the integers here). We call these maps slopes. Slopes form a group with the natural addition $(s+r)(n) = s(n) + r(n)$. The maps such that the set $s(Z)$ is finite (finite range functions) form a subgroup of slopes. The additive group of real numbers is defined as the quotient group of slopes by the (sub)group of finite range functions. The multiplication is defined as the projection of the composition of slopes into the resulting quotient (coset) space.

62.1 The definition of real numbers

This section contains the construction of the ring of real numbers as classes of slopes - integer almost homomorphisms. The real definitions are in `Group_ZF_2` theory, here we just specialize the definitions of almost homomorphisms, their equivalence and operations to the additive group of integers from the general case of abelian groups considered in Group_ZF_2.

The set of slopes is defined as the set of almost homomorphisms on the additive group of integers.

definition
```
Slopes ≡ AlmostHoms(int,IntegerAddition)
```

The first operation on slopes (pointwise addition) is a special case of the first operation on almost homomorphisms.

definition
```
SlopeOp1 ≡ AlHomOp1(int,IntegerAddition)
```

The second operation on slopes (composition) is a special case of the second operation on almost homomorphisms.

definition
```
SlopeOp2 ≡ AlHomOp2(int,IntegerAddition)
```

Bounded integer maps are functions from integers to integers that have finite range. They play a role of zero in the set of real numbers we are constructing.

definition
```
BoundedIntMaps ≡ FinRangeFunctions(int,int)
```

Bounded integer maps form a normal subgroup of slopes. The equivalence relation on slopes is the (group) quotient relation defined by this subgroup.

definition
```
SlopeEquivalenceRel ≡ QuotientGroupRel(Slopes,SlopeOp1,BoundedIntMaps)
```

The set of real numbers is the set of equivalence classes of slopes.

definition
```
RealNumbers ≡ Slopes//SlopeEquivalenceRel
```

The addition on real numbers is defined as the projection of pointwise addition of slopes on the quotient. This means that the additive group of real numbers is the quotient group: the group of slopes (with pointwise addition) defined by the normal subgroup of bounded integer maps.

definition
```
RealAddition ≡ ProjFun2(Slopes,SlopeEquivalenceRel,SlopeOp1)
```

Multiplication is defined as the projection of composition of slopes on the quotient. The fact that it works is probably the most surprising part of the construction.

definition
```
RealMultiplication ≡ ProjFun2(Slopes,SlopeEquivalenceRel,SlopeOp2)
```

We first show that we can use theorems proven in some proof contexts (locales). The locale group1 requires assumption that we deal with an abelian group. The next lemma allows to use all theorems proven in the context called group1.

```
lemma Real_ZF_1_L1: shows group1(int,IntegerAddition)
  using group1_axioms.intro group1_def Int_ZF_1_T2 by simp
```

Real numbers form a ring. This is a special case of the theorem proven in Ring_ZF_1.thy, where we show the same in general for almost homomorphisms rather than slopes.

```
theorem Real_ZF_1_T1: shows IsAring(RealNumbers,RealAddition,RealMultiplication)
proof -
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let Op2 = AlHomOp2(int,IntegerAddition)
  let R = QuotientGroupRel(AH,Op1,FR)
  let A = ProjFun2(AH,R,Op1)
  let M = ProjFun2(AH,R,Op2)
  have IsAring(AH//R,A,M) using Real_ZF_1_L1 group1.Ring_ZF_1_1_T1
    by simp
  then show thesis using Slopes_def SlopeOp2_def SlopeOp1_def
    BoundedIntMaps_def SlopeEquivalenceRel_def RealNumbers_def
    RealAddition_def RealMultiplication_def by simp
qed
```

We can use theorems proven in group0 and group1 contexts applied to the group of real numbers.

```
lemma Real_ZF_1_L2: shows
  group0(RealNumbers,RealAddition)
  RealAddition {is commutative on} RealNumbers
  group1(RealNumbers,RealAddition)
proof -
  have
    IsAgroup(RealNumbers,RealAddition)
    RealAddition {is commutative on} RealNumbers
    using Real_ZF_1_T1 IsAring_def by auto
  then show
    group0(RealNumbers,RealAddition)
    RealAddition {is commutative on} RealNumbers
    group1(RealNumbers,RealAddition)
    using group1_axioms.intro group0_def group1_def
    by auto
qed
```

Let's define some notation.

```
locale real0 =

  fixes real (ℝ)
  defines real_def [simp]: ℝ ≡ RealNumbers
```

```
  fixes ra (infixl + 69)
  defines ra_def [simp]: a+ b ≡ RealAddition⟨a,b⟩

  fixes rminus (- _ 72)
  defines rminus_def [simp]:-a ≡ GroupInv(ℝ,RealAddition)(a)

  fixes rsub (infixl - 69)
  defines rsub_def [simp]: a-b ≡ a+(-b)

  fixes rm (infixl · 70)
  defines rm_def [simp]: a·b ≡ RealMultiplication⟨a,b⟩

  fixes rzero (0)
  defines rzero_def [simp]:
  0 ≡ TheNeutralElement(RealNumbers,RealAddition)

  fixes rone (1)
  defines rone_def [simp]:
  1 ≡ TheNeutralElement(RealNumbers,RealMultiplication)

  fixes rtwo (2)
  defines rtwo_def [simp]: 2  ≡ 1+1

  fixes non_zero (ℝ0)
  defines non_zero_def[simp]: ℝ0 ≡ ℝ-{0}

  fixes inv (_⁻¹  [90] 91)
  defines inv_def[simp]:
  a⁻¹ ≡ GroupInv(ℝ0,restrict(RealMultiplication,ℝ0×ℝ0))(a)
```

In `real0` context all theorems proven in the `ring0`, context are valid.

```
lemma (in real0) Real_ZF_1_L3: shows
  ring0(ℝ,RealAddition,RealMultiplication)
  using Real_ZF_1_T1 ring0_def ring0.Ring_ZF_1_L1
  by auto
```

Lets try out our notation to see that zero and one are real numbers.

```
lemma (in real0) Real_ZF_1_L4: shows 0∈ℝ  1∈ℝ
  using Real_ZF_1_L3 ring0.Ring_ZF_1_L2 by auto
```

The lemma below lists some properties that require one real number to state.

```
lemma (in real0) Real_ZF_1_L5: assumes A1: a∈ℝ
  shows
  (-a) ∈ ℝ
  (-(-a)) = a
  a+0 = a
  0+a = a
  a·1 = a
```

```
  1·a = a
  a-a = 0
  a-0 = a
  using assms Real_ZF_1_L3 ring0.Ring_ZF_1_L3 by auto
```

The lemma below lists some properties that require two real numbers to state.

```
lemma (in real0) Real_ZF_1_L6: assumes a∈ℝ  b∈ℝ
  shows
  a+b ∈ ℝ
  a-b ∈ ℝ
  a·b ∈ ℝ
  a+b = b+a
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  using assms Real_ZF_1_L3 ring0.Ring_ZF_1_L4 ring0.Ring_ZF_1_L7
  by auto
```

Multiplication of reals is associative.

```
lemma (in real0) Real_ZF_1_L6A: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows a·(b·c) = (a·b)·c
  using assms Real_ZF_1_L3 ring0.Ring_ZF_1_L11
  by simp
```

Addition is distributive with respect to multiplication.

```
lemma (in real0) Real_ZF_1_L7: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows
  a·(b+c) = a·b + a·c
  (b+c)·a = b·a + c·a
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using assms Real_ZF_1_L3 ring0.ring_oper_distr  ring0.Ring_ZF_1_L8
  by auto
```

A simple rearrangement with four real numbers.

```
lemma (in real0) Real_ZF_1_L7A:
  assumes a∈ℝ  b∈ℝ  c∈ℝ  d∈ℝ
  shows a-b + (c-d) = a+c-b-d
  using assms Real_ZF_1_L2 group0.group0_4_L8A by simp
```

`RealAddition` is defined as the projection of the first operation on slopes (that is, slope addition) on the quotient (slopes divided by the "almost equal" relation. The next lemma plays with definitions to show that this is the same as the operation induced on the appriopriate quotient group. The names `AH`, `Op1` and `FR` are used in `group1` context to denote almost homomorphisms, the first operation on `AH` and finite range functions resp.

```
lemma Real_ZF_1_L8: assumes
```

```
  AH = AlmostHoms(int,IntegerAddition) and
  Op1 = AlHomOp1(int,IntegerAddition) and
  FR = FinRangeFunctions(int,int)
  shows RealAddition = QuotientGroupOp(AH,Op1,FR)
  using assms RealAddition_def SlopeEquivalenceRel_def
    QuotientGroupOp_def Slopes_def SlopeOp1_def BoundedIntMaps_def
  by simp
```

The symbol **0** in the `real0` context is defined as the neutral element of real addition. The next lemma shows that this is the same as the neutral element of the appriopriate quotient group.

```
lemma (in real0) Real_ZF_1_L9: assumes
  AH = AlmostHoms(int,IntegerAddition) and
  Op1 = AlHomOp1(int,IntegerAddition) and
  FR = FinRangeFunctions(int,int) and
  r = QuotientGroupRel(AH,Op1,FR)
  shows
  TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR)) = 0
  SlopeEquivalenceRel = r
  using assms Slopes_def Real_ZF_1_L8 RealNumbers_def
    SlopeEquivalenceRel_def SlopeOp1_def BoundedIntMaps_def
  by auto
```

Zero is the class of any finite range function.

```
lemma (in real0) Real_ZF_1_L10:
  assumes A1: s ∈ Slopes
  shows SlopeEquivalenceRel{s} = 0 ⟷ s ∈ BoundedIntMaps
proof -
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let r = QuotientGroupRel(AH,Op1,FR)
  let e = TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR))
  from A1 have
    group1(int,IntegerAddition)
    s∈AH
    using Real_ZF_1_L1 Slopes_def
    by auto
  then have r{s} = e ⟷ s ∈ FR
    using group1.Group_ZF_3_3_L5 by simp
  moreover have
    r = SlopeEquivalenceRel
    e = 0
    FR = BoundedIntMaps
    using SlopeEquivalenceRel_def Slopes_def SlopeOp1_def
      BoundedIntMaps_def Real_ZF_1_L9 by auto
  ultimately show thesis by simp
qed
```

We will need a couple of results from `Group_ZF_3.thy` The first two that state that the definition of addition and multiplication of real numbers are consistent, that is the result does not depend on the choice of the slopes representing the numbers. The second one implies that what we call `SlopeEquivalenceRel` is actually an equivalence relation on the set of slopes. We also show that the neutral element of the multiplicative operation on reals (in short number 1) is the class of the identity function on integers.

```
lemma Real_ZF_1_L11: shows
  Congruent2(SlopeEquivalenceRel,SlopeOp1)
  Congruent2(SlopeEquivalenceRel,SlopeOp2)
  SlopeEquivalenceRel ⊆ Slopes × Slopes
  equiv(Slopes, SlopeEquivalenceRel)
  SlopeEquivalenceRel{id(int)} =
  TheNeutralElement(RealNumbers,RealMultiplication)
  BoundedIntMaps ⊆ Slopes
proof -
  let G = int
  let f = IntegerAddition
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let Op2 = AlHomOp2(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let R = QuotientGroupRel(AH,Op1,FR)
   have
     Congruent2(R,Op1)
     Congruent2(R,Op2)
    using Real_ZF_1_L1 group1.Group_ZF_3_4_L13A group1.Group_ZF_3_3_L4
    by auto
  then show
    Congruent2(SlopeEquivalenceRel,SlopeOp1)
    Congruent2(SlopeEquivalenceRel,SlopeOp2)
    using SlopeEquivalenceRel_def SlopeOp1_def Slopes_def
      BoundedIntMaps_def SlopeOp2_def by auto
  have equiv(AH,R)
    using Real_ZF_1_L1 group1.Group_ZF_3_3_L3 by simp
  then show equiv(Slopes,SlopeEquivalenceRel)
    using BoundedIntMaps_def SlopeEquivalenceRel_def SlopeOp1_def Slopes_def
    by simp
  then show SlopeEquivalenceRel ⊆ Slopes × Slopes
    using equiv_type by simp
  have R{id(int)} = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
    using Real_ZF_1_L1 group1.Group_ZF_3_4_T2 by simp
  then show  SlopeEquivalenceRel{id(int)} =
    TheNeutralElement(RealNumbers,RealMultiplication)
    using Slopes_def RealNumbers_def
    SlopeEquivalenceRel_def SlopeOp1_def BoundedIntMaps_def
    RealMultiplication_def SlopeOp2_def
    by simp
```

```
  have FR ⊆ AH using Real_ZF_1_L1 group1.Group_ZF_3_3_L1
    by simp
  then show BoundedIntMaps ⊆ Slopes
    using BoundedIntMaps_def Slopes_def by simp
qed
```

A one-side implication of the equivalence from `Real_ZF_1_L10`: the class of a bounded integer map is the real zero.

```
lemma (in real0) Real_ZF_1_L11A: assumes s ∈ BoundedIntMaps
  shows SlopeEquivalenceRel{s} = 0
  using assms Real_ZF_1_L11 Real_ZF_1_L10 by auto
```

The next lemma is rephrases the result from `Group_ZF_3.thy` that says that the negative (the group inverse with respect to real addition) of the class of a slope is the class of that slope composed with the integer additive group inverse. The result and proof is not very readable as we use mostly generic set theory notation with long names here. `Real_ZF_1.thy` contains the same statement written in a more readable notation: $[-s] = -[s]$.

```
lemma (in real0) Real_ZF_1_L12: assumes A1: s ∈ Slopes and
  Dr: r = QuotientGroupRel(Slopes,SlopeOp1,BoundedIntMaps)
  shows r{GroupInv(int,IntegerAddition) O s} = -(r{s})
proof -
  let G = int
  let f = IntegerAddition
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let F = ProjFun2(Slopes,r,SlopeOp1)
  from A1 Dr have
    group1(G, f)
    s ∈ AlmostHoms(G, f)
    r = QuotientGroupRel(
    AlmostHoms(G, f), AlHomOp1(G, f), FinRangeFunctions(G, G))
    and F = ProjFun2(AlmostHoms(G, f), r, AlHomOp1(G, f))
    using Real_ZF_1_L1 Slopes_def SlopeOp1_def BoundedIntMaps_def
    by auto
  then have
    r{GroupInv(G, f) O s} =
    GroupInv(AlmostHoms(G, f) // r, F)(r  {s})
    using group1.Group_ZF_3_3_L6 by simp
  with Dr show thesis
    using RealNumbers_def Slopes_def SlopeEquivalenceRel_def RealAddition_def
    by simp
qed
```

Two classes are equal iff the slopes that represent them are almost equal.

```
lemma Real_ZF_1_L13: assumes s ∈ Slopes  p ∈ Slopes
  and r = SlopeEquivalenceRel
```

```
  shows r{s} = r{p} ⟷ ⟨s,p⟩ ∈  r
  using assms Real_ZF_1_L11 eq_equiv_class equiv_class_eq
  by blast
```

Identity function on integers is a slope. This lemma concludes the easy part of the construction that follows from the fact that slope equivalence classes form a ring. It is easy to see that multiplication of classes of almost homomorphisms is not commutative in general. The remaining properties of real numbers, like commutativity of multiplication and the existence of multiplicative inverses have to be proven using properties of the group of integers, rather that in general setting of abelian groups. This is done in the `Real_ZF_1` theory.

```
lemma Real_ZF_1_L14: shows id(int) ∈ Slopes
proof -
  have id(int) ∈ AlmostHoms(int,IntegerAddition)
    using Real_ZF_1_L1 group1.Group_ZF_3_4_L15
    by simp
  then show thesis using Slopes_def by simp
qed

end
```

63 Construction of real numbers

```
theory Real_ZF_1 imports Real_ZF Int_ZF_3 OrderedField_ZF

begin
```

In this theory file we continue the construction of real numbers started in `Real_ZF` to a succesful conclusion. We put here those parts of the construction that can not be done in the general settings of abelian groups and require integers.

63.1 Definitions and notation

In this section we define notions and notation needed for the rest of the construction.

We define positive slopes as those that take an infinite number of posititive values on the positive integers (see `Int_ZF_2` for properties of positive slopes).

```
definition
  PositiveSlopes ≡ {s ∈ Slopes.
  s(PositiveIntegers) ∩  PositiveIntegers ∉ Fin(int)}
```

The order on the set of real numbers is constructed by specifying the set of positive reals. This set is defined as the projection of the set of positive slopes.

definition
`PositiveReals` $\equiv$ `{SlopeEquivalenceRel{s}. s` $\in$ `PositiveSlopes}`

The order relation on real numbers is constructed from the set of positive elements in a standard way (see section "Alternative definitions" in `OrderedGroup_ZF`.)

definition
`OrderOnReals` $\equiv$ `OrderFromPosSet(RealNumbers,RealAddition,PositiveReals)`

The next locale extends the locale `real0` to define notation specific to the construction of real numbers. The notation follows the one defined in `Int_ZF_2.thy`. If m is an integer, then the real number which is the class of the slope $n \mapsto m \cdot n$ is denoted $\mathtt{m}^R$. For a real number a notation $\lfloor a \rfloor$ means the largest integer m such that the real version of it (that is, m^R) is not greater than a. For an integer m and a subset of reals S the expression $\Gamma(S, m)$ is defined as $\max\{\lfloor p^R \cdot x \rfloor : x \in S\}$. This is plays a role in the proof of completeness of real numbers. We also reuse some notation defined in the `int0` context, like $\mathbb{Z}_+$ (the set of positive integers) and $\text{abs}(m)$ (the absolute value of an integer, and some defined in the `int1` context, like the addition (`+`) and composition ($\circ$ of slopes.

locale `real1 = real0 +`

fixes `AlEq` (**infix** $\sim$ `68`)
defines `AlEq_def[simp]: s` $\sim$ `r` $\equiv$ $\langle$`s,r`$\rangle$ $\in$ `SlopeEquivalenceRel`

fixes `slope_add` (**infix** `+ 70`)
defines `slope_add_def[simp]:`
`s + r` $\equiv$ `SlopeOp1`$\langle$`s,r`$\rangle$

fixes `slope_comp` (**infix** $\circ$ `71`)
defines `slope_comp_def[simp]: s` $\circ$ `r` $\equiv$ `SlopeOp2`$\langle$`s,r`$\rangle$

fixes `slopes` ($\mathcal{S}$)
defines `slopes_def[simp]:` $\mathcal{S}$ $\equiv$ `AlmostHoms(int,IntegerAddition)`

fixes `posslopes` ($\mathcal{S}_+$)
defines `posslopes_def[simp]:` $\mathcal{S}_+$ $\equiv$ `PositiveSlopes`

fixes `slope_class` (`[ _ ]`)
defines `slope_class_def[simp]: [f]` $\equiv$ `SlopeEquivalenceRel{f}`

fixes `slope_neg (-_ [90] 91)`
defines `slope_neg_def[simp]: -s` $\equiv$ `GroupInv(int,IntegerAddition) O s`

fixes `lesseqr` (**infix** $\leq$ `60`)
defines `lesseqr_def[simp]: a` $\leq$ `b` $\equiv$ $\langle$`a,b`$\rangle$ $\in$ `OrderOnReals`

fixes sless (**infix** < 60)
defines sless_def[simp]: $a < b \equiv a\leq b \wedge a\neq b$

fixes positivereals ($\mathbb{R}_+$)
defines positivereals_def[simp]: $\mathbb{R}_+ \equiv$ PositiveSet($\mathbb{R}$,RealAddition,OrderOnReals)

fixes intembed (_^{R} [90] 91)
defines intembed_def[simp]:
$m^R \equiv$ [{$\langle$n,IntegerMultiplication$\langle$m,n$\rangle$ $\rangle$. n $\in$ int}]

fixes floor ($\lfloor$ _ $\rfloor$)
defines floor_def[simp]:
$\lfloor a \rfloor \equiv$ Maximum(IntegerOrder,{m $\in$ int. $m^R \leq$ a})

fixes Γ
defines Γ_def[simp]: Γ(S,p) $\equiv$ Maximum(IntegerOrder,{$\lfloor p^R\cdot x\rfloor$. x$\in$S})

fixes ia (**infixl** + 69)
defines ia_def[simp]: a+b $\equiv$ IntegerAddition$\langle$ a,b$\rangle$

fixes iminus (- _ 72)
defines iminus_def[simp]: -a $\equiv$ GroupInv(int,IntegerAddition)(a)

fixes isub (**infixl** - 69)
defines isub_def[simp]: a-b $\equiv$ a+ (- b)

fixes intpositives ($\mathbb{Z}_+$)
defines intpositives_def[simp]:
$\mathbb{Z}_+ \equiv$ PositiveSet(int,IntegerAddition,IntegerOrder)

fixes zlesseq (**infix** $\leq$ 60)
defines lesseq_def[simp]: m $\leq$ n $\equiv$ $\langle$m,n$\rangle$ $\in$ IntegerOrder

fixes imult (**infixl** $\cdot$ 70)
defines imult_def[simp]: a$\cdot$b $\equiv$ IntegerMultiplication$\langle$ a,b$\rangle$

fixes izero ($\mathbf{0}_Z$)
defines izero_def[simp]: $\mathbf{0}_Z \equiv$ TheNeutralElement(int,IntegerAddition)

fixes ione ($\mathbf{1}_Z$)
defines ione_def[simp]: $\mathbf{1}_Z \equiv$ TheNeutralElement(int,IntegerMultiplication)

fixes itwo ($\mathbf{2}_Z$)
defines itwo_def[simp]: $\mathbf{2}_Z \equiv \mathbf{1}_Z$+$\mathbf{1}_Z$

fixes abs
defines abs_def[simp]:
abs(m) $\equiv$ AbsoluteValue(int,IntegerAddition,IntegerOrder)(m)

```
  fixes δ
  defines δ_def[simp]: δ(s,m,n) ≡ s(m+n)-s(m)-s(n)
```

63.2 Multiplication of real numbers

Multiplication of real numbers is defined as a projection of composition of slopes onto the space of equivalence classes of slopes. Thus, the product of the real numbers given as classes of slopes s and r is defined as the class of $s \circ r$. The goal of this section is to show that multiplication defined this way is commutative.

Let's recall a theorem from `Int_ZF_2.thy` that states that if f, g are slopes, then $f \circ g$ is equivalent to $g \circ f$. Here we conclude from that that the classes of $f \circ g$ and $g \circ f$ are the same.

```
lemma (in real1) Real_ZF_1_1_L2: assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows [f∘g] = [g∘f]
proof -
  from A1 have f∘g ∼ g∘f
    using Slopes_def int1.Arthan_Th_9 SlopeOp1_def BoundedIntMaps_def
      SlopeEquivalenceRel_def SlopeOp2_def by simp
  then show thesis using Real_ZF_1_L11 equiv_class_eq
    by simp
qed
```

Classes of slopes are real numbers.

```
lemma (in real1) Real_ZF_1_1_L3: assumes A1: f ∈ 𝒮
  shows [f] ∈ ℝ
proof -
  from A1 have [f] ∈ Slopes//SlopeEquivalenceRel
    using Slopes_def quotientI by simp
  then show [f] ∈ ℝ using RealNumbers_def by simp
qed
```

Each real number is a class of a slope.

```
lemma (in real1) Real_ZF_1_1_L3A: assumes A1: a∈ℝ
  shows ∃f∈𝒮 . a = [f]
proof -
  from A1 have a ∈ 𝒮//SlopeEquivalenceRel
    using RealNumbers_def Slopes_def by simp
  then show thesis using quotient_def
    by simp
qed
```

It is useful to have the definition of addition and multiplication in the `real1` context notation.

```
lemma (in real1) Real_ZF_1_1_L4:
```

```
  assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows
  [f] + [g] = [f+g]
  [f] · [g] = [f∘g]
proof -
  let r = SlopeEquivalenceRel
  have [f]·[g] = ProjFun2(𝒮,r,SlopeOp2)⟨[f],[g]⟩
    using RealMultiplication_def Slopes_def by simp
  also from A1 have ... = [f∘g]
    using Real_ZF_1_L11 EquivClass_1_L10 Slopes_def
    by simp
  finally show [f] · [g] = [f∘g] by simp
  have [f] + [g] = ProjFun2(𝒮,r,SlopeOp1)⟨[f],[g]⟩
    using RealAddition_def Slopes_def by simp
  also from A1 have ... = [f+g]
    using Real_ZF_1_L11 EquivClass_1_L10 Slopes_def
    by simp
  finally show [f] + [g] = [f+g] by simp
qed
```

The next lemma is essentially the same as Real_ZF_1_L12, but written in the notation defined in the real1 context. It states that if f is a slope, then $-[f] = [-f]$.

```
lemma (in real1) Real_ZF_1_1_L4A: assumes f ∈ 𝒮
  shows [-f] = -[f]
  using assms Slopes_def SlopeEquivalenceRel_def Real_ZF_1_L12
  by simp
```

Subtracting real numbers correspods to adding the opposite slope.

```
lemma (in real1) Real_ZF_1_1_L4B: assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows [f] - [g] = [f+(-g)]
proof -
  from A1 have [f+(-g)] = [f] + [-g]
    using Slopes_def BoundedIntMaps_def int1.Int_ZF_2_1_L12
      Real_ZF_1_1_L4 by simp
  with A1 show [f] - [g] = [f+(-g)]
    using Real_ZF_1_1_L4A by simp
qed
```

Multiplication of real numbers is commutative.

```
theorem (in real1) real_mult_commute: assumes A1: a∈ℝ  b∈ℝ
  shows a·b = b·a
proof -
  from A1 have
    ∃f∈𝒮 . a = [f]
    ∃g∈𝒮 . b = [g]
    using Real_ZF_1_1_L3A by auto
  then obtain f g where
```

```
    f ∈ 𝒮   g ∈ 𝒮 and a = [f]  b = [g]
    by auto
  then show a·b = b·a
    using Real_ZF_1_1_L4 Real_ZF_1_1_L2 by simp
qed
```

Multiplication is commutative on reals.

```
lemma real_mult_commutative: shows
  RealMultiplication {is commutative on} RealNumbers
  using real1.real_mult_commute IsCommutative_def
  by simp
```

The neutral element of multiplication of reals (denoted as **1** in the `real1` context) is the class of identity function on integers. This is really shown in `Real_ZF_1_L11`, here we only rewrite it in the notation used in the `real1` context.

```
lemma (in real1) real_one_cl_identity: shows [id(int)] = 1
  using Real_ZF_1_L11 by simp
```

If f is bounded, then its class is the neutral element of additive operation on reals (denoted as **0** in the `real1` context).

```
lemma (in real1) real_zero_cl_bounded_map:
  assumes f ∈ BoundedIntMaps shows [f] = 0
  using assms Real_ZF_1_L11A by simp
```

Two real numbers are equal iff the slopes that represent them are almost equal. This is proven in `Real_ZF_1_L13`, here we just rewrite it in the notation used in the `real1` context.

```
lemma (in real1) Real_ZF_1_1_L5:
  assumes f ∈ 𝒮   g ∈ 𝒮
  shows [f] = [g] ⟷ f ∼ g
  using assms Slopes_def Real_ZF_1_L13 by simp
```

If the pair of function belongs to the slope equivalence relation, then their classes are equal. This is convenient, because we don't need to assume that f, g are slopes (follows from the fact that $f \sim g$).

```
lemma (in real1) Real_ZF_1_1_L5A: assumes f ∼ g
  shows [f] = [g]
  using assms Real_ZF_1_L11 Slopes_def Real_ZF_1_1_L5
  by auto
```

Identity function on integers is a slope. This is proven in `Real_ZF_1_L13`, here we just rewrite it in the notation used in the `real1` context.

```
lemma (in real1) id_on_int_is_slope: shows id(int) ∈ 𝒮
  using Real_ZF_1_L14 Slopes_def by simp
```

A result from `Int_ZF_2.thy`: the identity function on integers is not almost equal to any bounded function.

```
lemma (in real1) Real_ZF_1_1_L7:
  assumes A1: f ∈ BoundedIntMaps
  shows ¬(id(int) ∼ f)
  using assms Slopes_def SlopeOp1_def BoundedIntMaps_def
    SlopeEquivalenceRel_def BoundedIntMaps_def int1.Int_ZF_2_3_L12
  by simp
```

Zero is not one.

```
lemma (in real1) real_zero_not_one: shows 1≠0
proof -
  { assume A1: 1=0
    have ∃f ∈ 𝒮. 0 = [f]
      using  Real_ZF_1_L4 Real_ZF_1_1_L3A by simp
    with A1 have
      ∃f ∈ 𝒮. [id(int)] = [f] ∧ [f] = 0
      using real_one_cl_identity by auto
    then have False using Real_ZF_1_1_L5 Slopes_def
      Real_ZF_1_L10 Real_ZF_1_1_L7 id_on_int_is_slope
      by auto
  } then show 1≠0 by auto
qed
```

Negative of a real number is a real number. Property of groups.

```
lemma (in real1) Real_ZF_1_1_L8: assumes a∈ℝ shows (-a) ∈ ℝ
  using assms Real_ZF_1_L2 group0.inverse_in_group
  by simp
```

An identity with three real numbers.

```
lemma (in real1) Real_ZF_1_1_L9: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows a·(b·c) = a·c·b
  using assms real_mult_commutative Real_ZF_1_L3 ring0.Ring_ZF_2_L4
  by simp
```

63.3 The order on reals

In this section we show that the order relation defined by prescribing the set of positive reals as the projection of the set of positive slopes makes the ring of real numbers into an ordered ring. We also collect the facts about ordered groups and rings that we use in the construction.

Positive slopes are slopes and positive reals are real.

```
lemma Real_ZF_1_2_L1: shows
  PositiveSlopes ⊆ Slopes
  PositiveReals ⊆ RealNumbers
proof -
  have PositiveSlopes =
    {s ∈ Slopes. s(PositiveIntegers) ∩ PositiveIntegers ∉ Fin(int)}
    using PositiveSlopes_def by simp
```

```
  then show PositiveSlopes ⊆ Slopes by (rule subset_with_property)
  then have
    {SlopeEquivalenceRel{s}. s ∈ PositiveSlopes } ⊆
    Slopes//SlopeEquivalenceRel
    using EquivClass_1_L1A by simp
  then show PositiveReals ⊆ RealNumbers
    using PositiveReals_def RealNumbers_def by simp
qed
```

Positive reals are the same as classes of a positive slopes.

```
lemma (in real1) Real_ZF_1_2_L2:
  shows a ∈ PositiveReals ⟷ (∃f∈𝒮₊. a = [f])
proof
  assume a ∈ PositiveReals
  then have a ∈ {([s]). s ∈ 𝒮₊} using PositiveReals_def
    by simp
  then show ∃f∈𝒮₊. a = [f] by auto
next assume ∃f∈𝒮₊. a = [f]
  then have  a ∈ {([s]). s ∈ 𝒮₊} by auto
  then show a ∈ PositiveReals using PositiveReals_def
    by simp
qed
```

Let's recall from `Int_ZF_2.thy` that the sum and composition of positive slopes is a positive slope.

```
lemma (in real1) Real_ZF_1_2_L3:
  assumes f∈𝒮₊  g∈𝒮₊
  shows
  f+g ∈ 𝒮₊
  f∘g ∈ 𝒮₊
  using assms Slopes_def PositiveSlopes_def PositiveIntegers_def
    SlopeOp1_def int1.sum_of_pos_sls_is_pos_sl
    SlopeOp2_def int1.comp_of_pos_sls_is_pos_sl
  by auto
```

Bounded integer maps are not positive slopes.

```
lemma (in real1) Real_ZF_1_2_L5:
  assumes f ∈ BoundedIntMaps
  shows f ∉ 𝒮₊
  using assms BoundedIntMaps_def Slopes_def PositiveSlopes_def
    PositiveIntegers_def int1.Int_ZF_2_3_L1B by simp
```

The set of positive reals is closed under addition and multiplication. Zero (the neutral element of addition) is not a positive number.

```
lemma (in real1) Real_ZF_1_2_L6: shows
  PositiveReals {is closed under} RealAddition
  PositiveReals {is closed under} RealMultiplication
  0 ∉ PositiveReals
```

proof -
  { **fix** a **fix** b
    assume a ∈ PositiveReals **and** b ∈ PositiveReals
    then obtain f g **where**
      I: f ∈ $\mathcal{S}_+$ g ∈ $\mathcal{S}_+$ **and**
      II: a = [f] b = [g]
      using Real_ZF_1_2_L2 **by** auto
    then have f ∈ $\mathcal{S}$ g ∈ $\mathcal{S}$ **using** Real_ZF_1_2_L1 Slopes_def
      by auto
    with I II **have**
      a+b ∈ PositiveReals ∧ a·b ∈ PositiveReals
       using Real_ZF_1_1_L4 Real_ZF_1_2_L3 Real_ZF_1_2_L2
       by auto
  } **then show**
      PositiveReals {is closed under} RealAddition
      PositiveReals {is closed under} RealMultiplication
    using IsOpClosed_def
    by auto
  { **assume** **0** ∈ PositiveReals
    then obtain f **where** f ∈ $\mathcal{S}_+$ **and** **0** = [f]
      using Real_ZF_1_2_L2 **by** auto
    then have False
      using Real_ZF_1_2_L1 Slopes_def Real_ZF_1_L10 Real_ZF_1_2_L5
      by auto
  } **then show** **0** ∉ PositiveReals **by** auto
qed

If a class of a slope f is not zero, then either f is a positive slope or $-f$ is a positive slope. The real proof is in Int_ZF_2.thy.

lemma (**in** real1) Real_ZF_1_2_L7:
  assumes A1: f ∈ $\mathcal{S}$ **and** A2: [f] ≠ **0**
  shows (f ∈ $\mathcal{S}_+$) Xor ((-f) ∈ $\mathcal{S}_+$)
  using assms Slopes_def SlopeEquivalenceRel_def BoundedIntMaps_def
    PositiveSlopes_def PositiveIntegers_def
    Real_ZF_1_L10 int1.Int_ZF_2_3_L8 **by** simp

The next lemma rephrases Int_ZF_2_3_L10 in the notation used in real1 context.

lemma (**in** real1) Real_ZF_1_2_L8:
  assumes A1: f ∈ $\mathcal{S}$ g ∈ $\mathcal{S}$
  and A2: (f ∈ $\mathcal{S}_+$) Xor (g ∈ $\mathcal{S}_+$)
  shows ([f] ∈ PositiveReals) Xor ([g] ∈ PositiveReals)
  using assms PositiveReals_def SlopeEquivalenceRel_def Slopes_def
    SlopeOp1_def BoundedIntMaps_def PositiveSlopes_def PositiveIntegers_def
    int1.Int_ZF_2_3_L10 **by** simp

The trichotomy law for the (potential) order on reals: if $a \neq 0$, then either a is positive or $-a$ is potitive.

lemma (**in** real1) Real_ZF_1_2_L9:

```
  assumes A1: a∈ℝ and A2: a≠0
  shows (a ∈ PositiveReals) Xor ((-a) ∈ PositiveReals)
proof -
  from A1 obtain f where I: f ∈ 𝒮  a = [f]
    using Real_ZF_1_1_L3A by auto
  with A2 have ([f] ∈ PositiveReals) Xor ([-f] ∈ PositiveReals)
    using Slopes_def BoundedIntMaps_def int1.Int_ZF_2_1_L12
      Real_ZF_1_2_L7 Real_ZF_1_2_L8 by simp
  with I show (a ∈ PositiveReals) Xor ((-a) ∈ PositiveReals)
    using Real_ZF_1_1_L4A by simp
qed
```

Finally we are ready to prove that real numbers form an ordered ring with no zero divisors.

```
theorem reals_are_ord_ring: shows
  IsAnOrdRing(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  OrderOnReals {is total on} RealNumbers
  PositiveSet(RealNumbers,RealAddition,OrderOnReals) = PositiveReals
  HasNoZeroDivs(RealNumbers,RealAddition,RealMultiplication)
proof -
  let R = RealNumbers
  let A = RealAddition
  let M = RealMultiplication
  let P = PositiveReals
  let r = OrderOnReals
  let z = TheNeutralElement(R, A)
  have I:
    ring0(R, A, M)
    M {is commutative on} R
    P ⊆ R
    P {is closed under} A
    TheNeutralElement(R, A) ∉ P
    ∀a∈R. a ≠ z ⟶ (a ∈ P) Xor (GroupInv(R, A)(a) ∈ P)
    P {is closed under} M
    r = OrderFromPosSet(R, A, P)
    using real0.Real_ZF_1_L3 real_mult_commutative Real_ZF_1_2_L1
      real1.Real_ZF_1_2_L6 real1.Real_ZF_1_2_L9 OrderOnReals_def
    by auto
  then show IsAnOrdRing(R, A, M, r)
    by (rule ring0.ring_ord_by_positive_set)
  from I show r {is total on} R
    by (rule ring0.ring_ord_by_positive_set)
  from I show PositiveSet(R,A,r) = P
    by (rule ring0.ring_ord_by_positive_set)
  from I show HasNoZeroDivs(R,A,M)
    by (rule ring0.ring_ord_by_positive_set)
qed
```

All theorems proven in the `ring1` (about ordered rings), `group3` (about ordered groups) and group1 (about groups) contexts are valid as applied to ordered real numbers with addition and (real) order.

```
lemma Real_ZF_1_2_L10: shows
  ring1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  IsAnOrdGroup(RealNumbers,RealAddition,OrderOnReals)
  group3(RealNumbers,RealAddition,OrderOnReals)
  OrderOnReals {is total on} RealNumbers
proof -
  show ring1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
    using reals_are_ord_ring OrdRing_ZF_1_L2 by simp
  then show
    IsAnOrdGroup(RealNumbers,RealAddition,OrderOnReals)
    group3(RealNumbers,RealAddition,OrderOnReals)
    OrderOnReals {is total on} RealNumbers
    using ring1.OrdRing_ZF_1_L4 by auto
qed
```

If $a = b$ or $b - a$ is positive, then a is less or equal b.

```
lemma (in real1) Real_ZF_1_2_L11: assumes A1: a∈ℝ  b∈ℝ and
  A3: a=b ∨ b-a ∈ PositiveReals
  shows a≤b
  using assms reals_are_ord_ring Real_ZF_1_2_L10
    group3.OrderedGroup_ZF_1_L30 by simp
```

A sufficient condition for two classes to be in the real order.

```
lemma (in real1) Real_ZF_1_2_L12: assumes A1: f ∈ 𝒮  g ∈ 𝒮 and
  A2: f∼g ∨ (g + (-f)) ∈ 𝒮₊
  shows [f] ≤ [g]
proof -
  from A1 A2 have [f] = [g] ∨ [g]-[f] ∈ PositiveReals
    using Real_ZF_1_1_L5A Real_ZF_1_2_L2 Real_ZF_1_1_L4B
    by auto
  with A1 show [f] ≤ [g] using  Real_ZF_1_1_L3 Real_ZF_1_2_L11
    by simp
qed
```

Taking negative on both sides reverses the inequality, a case with an inverse on one side. Property of ordered groups.

```
lemma (in real1) Real_ZF_1_2_L13:
  assumes A1: a∈ℝ and A2: (-a) ≤ b
  shows (-b) ≤ a
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L5AG
  by simp
```

Real order is antisymmetric.

```
lemma (in real1) real_ord_antisym:
  assumes A1: a≤b  b≤a shows a=b
proof -
```

```
  from A1 have
    group3(RealNumbers,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,a⟩ ∈ OrderOnReals
    using Real_ZF_1_2_L10 by auto
  then show a=b by (rule group3.group_order_antisym)
qed
```

Real order is transitive.

```
lemma (in real1) real_ord_transitive: assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    group3(RealNumbers,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,c⟩ ∈ OrderOnReals
    using Real_ZF_1_2_L10 by auto
  then have ⟨a,c⟩ ∈ OrderOnReals
    by (rule group3.Group_order_transitive)
  then show a≤c by simp
qed
```

We can multiply both sides of an inequality by a nonnegative real number.

```
lemma (in real1) Real_ZF_1_2_L14:
  assumes a≤b and 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
  using assms Real_ZF_1_2_L10 ring1.OrdRing_ZF_1_L9
  by auto
```

A special case of `Real_ZF_1_2_L14`: we can multiply an inequality by a real number.

```
lemma (in real1) Real_ZF_1_2_L14A:
  assumes A1: a≤b and A2: c∈ℝ+
  shows c·a ≤ c·b
  using assms Real_ZF_1_2_L10 ring1.OrdRing_ZF_1_L9A
  by simp
```

In the `real1` context notation $a \leq b$ implies that a and b are real numbers.

```
lemma (in real1) Real_ZF_1_2_L15: assumes a≤b shows a∈ℝ  b∈ℝ
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L4
  by auto
```

$a \leq b$ implies that $0 \leq b - a$.

```
lemma (in real1) Real_ZF_1_2_L16: assumes a≤b
  shows 0 ≤ b-a
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L12A
  by simp
```

A sum of nonnegative elements is nonnegative.

```
lemma (in real1) Real_ZF_1_2_L17: assumes 0≤a 0≤b
  shows 0 ≤ a+b
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L12
  by simp
```

We can add sides of two inequalities

```
lemma (in real1) Real_ZF_1_2_L18: assumes a≤b  c≤d
  shows a+c ≤ b+d
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L5B
  by simp
```

The order on real is reflexive.

```
lemma (in real1) real_ord_refl: assumes a∈ℝ shows a≤a
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L3
  by simp
```

We can add a real number to both sides of an inequality.

```
lemma (in real1) add_num_to_ineq: assumes a≤b and c∈ℝ
  shows a+c ≤ b+c
  using assms Real_ZF_1_2_L10 IsAnOrdGroup_def by simp
```

We can put a number on the other side of an inequality, changing its sign.

```
lemma (in real1) Real_ZF_1_2_L19:
  assumes a∈ℝ  b∈ℝ and c ≤ a+b
  shows c-b ≤ a
  using assms  Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L9C
  by simp
```

What happens when one real number is not greater or equal than another?

```
lemma (in real1) Real_ZF_1_2_L20: assumes a∈ℝ  b∈ℝ and ¬(a≤b)
  shows b < a
proof -
  from assms have I:
    group3(ℝ,RealAddition,OrderOnReals)
    OrderOnReals {is total on} ℝ
    a∈ℝ  b∈ℝ  ¬(⟨a,b⟩ ∈ OrderOnReals)
    using Real_ZF_1_2_L10 by auto
  then have ⟨b,a⟩ ∈ OrderOnReals
    by (rule group3.OrderedGroup_ZF_1_L8)
  then have b ≤ a by simp
  moreover from I have a≠b by (rule group3.OrderedGroup_ZF_1_L8)
  ultimately show b < a by auto
qed
```

We can put a number on the other side of an inequality, changing its sign, version with a minus.

```
lemma (in real1) Real_ZF_1_2_L21:
  assumes a∈ℝ  b∈ℝ and c ≤ a-b
```

```
  shows c+b ≤ a
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L5J
  by simp
```

The order on reals is a relation on reals.

```
lemma (in real1) Real_ZF_1_2_L22: shows OrderOnReals ⊆ ℝ×ℝ
  using Real_ZF_1_2_L10 IsAnOrdGroup_def
  by simp
```

A set that is bounded above in the sense defined by order on reals is a subset of real numbers.

```
lemma (in real1) Real_ZF_1_2_L23:
  assumes A1: IsBoundedAbove(A,OrderOnReals)
  shows A ⊆ ℝ
  using A1 Real_ZF_1_2_L22 Order_ZF_3_L1A
  by blast
```

Properties of the maximum of three real numbers.

```
lemma (in real1) Real_ZF_1_2_L24:
  assumes A1: a∈ℝ  b∈ℝ  c∈ℝ
  shows
  Maximum(OrderOnReals,{a,b,c}) ∈ {a,b,c}
  Maximum(OrderOnReals,{a,b,c}) ∈ ℝ
  a ≤ Maximum(OrderOnReals,{a,b,c})
  b ≤ Maximum(OrderOnReals,{a,b,c})
  c ≤ Maximum(OrderOnReals,{a,b,c})
proof -
  have IsLinOrder(ℝ,OrderOnReals)
    using Real_ZF_1_2_L10 group3.group_ord_total_is_lin
    by simp
  with A1 show
    Maximum(OrderOnReals,{a,b,c}) ∈ {a,b,c}
    Maximum(OrderOnReals,{a,b,c}) ∈ ℝ
    a ≤ Maximum(OrderOnReals,{a,b,c})
    b ≤ Maximum(OrderOnReals,{a,b,c})
    c ≤ Maximum(OrderOnReals,{a,b,c})
    using Finite_ZF_1_L2A by auto
qed
```

A form of transitivity for the order on reals.

```
lemma (in real1) real_strict_ord_transit:
  assumes A1: a≤b and A2: b<c
  shows a<c
proof -
  from A1 A2 have I:
    group3(ℝ,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,c⟩ ∈ OrderOnReals ∧ b≠c
    using Real_ZF_1_2_L10 by auto
```

```
  then have ⟨a,c⟩ ∈ OrderOnReals ∧ a≠c by (rule group3.group_strict_ord_transit)
  then show a<c by simp
qed
```

We can multiply a right hand side of an inequality between positive real numbers by a number that is greater than one.

```
lemma (in real1) Real_ZF_1_2_L25:
  assumes b ∈ ℝ+ and a≤b and 1<c
  shows a<b·c
  using assms reals_are_ord_ring Real_ZF_1_2_L10 ring1.OrdRing_ZF_3_L17
  by simp
```

We can move a real number to the other side of a strict inequality, changing its sign.

```
lemma (in real1) Real_ZF_1_2_L26:
  assumes a∈ℝ  b∈ℝ and  a-b < c
  shows a < c+b
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L12B
  by simp
```

Real order is translation invariant.

```
lemma (in real1) real_ord_transl_inv:
  assumes a≤b and c∈ℝ
  shows c+a ≤ c+b
  using assms Real_ZF_1_2_L10 IsAnOrdGroup_def
  by simp
```

It is convenient to have the transitivity of the order on integers in the notation specific to `real1` context. This may be confusing for the presentation readers: even though ≤ and ≤ are printed in the same way, they are different symbols in the source. In the `real1` context the former denotes inequality between integers, and the latter denotes inequality between real numbers (classes of slopes). The next lemma is about transitivity of the order relation on integers.

```
lemma (in real1) int_order_transitive:
  assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    ⟨a,b⟩ ∈ IntegerOrder and ⟨b,c⟩ ∈ IntegerOrder
    by auto
  then have ⟨a,c⟩ ∈ IntegerOrder
    by (rule Int_ZF_2_L5)
  then show a≤c by simp
qed
```

A property of nonempty subsets of real numbers that don't have a maximum: for any element we can find one that is (strictly) greater.

```
lemma (in real1) Real_ZF_1_2_L27:
  assumes A⊆ℝ and ¬HasAmaximum(OrderOnReals,A) and x∈A
  shows ∃y∈A. x<y
  using assms Real_ZF_1_2_L10 group3.OrderedGroup_ZF_2_L2B
  by simp
```

The next lemma shows what happens when one real number is not greater or equal than another.

```
lemma (in real1) Real_ZF_1_2_L28:
  assumes a∈ℝ  b∈ℝ and ¬(a≤b)
  shows b<a
proof -
  from assms have
    group3(ℝ,RealAddition,OrderOnReals)
    OrderOnReals {is total on} ℝ
    a∈ℝ  b∈ℝ  ⟨a,b⟩ ∉ OrderOnReals
    using Real_ZF_1_2_L10 by auto
  then have ⟨b,a⟩ ∈ OrderOnReals  ∧ b≠a
    by (rule group3.OrderedGroup_ZF_1_L8)
  then show b<a by simp
qed
```

If a real number is less than another, then the second one can not be less or equal that the first.

```
lemma (in real1) Real_ZF_1_2_L29:
  assumes a<b shows ¬(b≤a)
proof -
  from assms have
    group3(ℝ,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  a≠b
    using Real_ZF_1_2_L10 by auto
  then have ⟨b,a⟩ ∉ OrderOnReals
    by (rule group3.OrderedGroup_ZF_1_L8AA)
  then show ¬(b≤a) by simp
qed
```

63.4 Inverting reals

In this section we tackle the issue of existence of (multiplicative) inverses of real numbers and show that real numbers form an ordered field. We also restate here some facts specific to ordered fields that we need for the construction. The actual proofs of most of these facts can be found in `Field_ZF.thy` and `OrderedField_ZF.thy`

We rewrite the theorem from `Int_ZF_2.thy` that shows that for every positive slope we can find one that is almost equal and has an inverse.

```
lemma (in real1) pos_slopes_have_inv: assumes f ∈ 𝒮₊
  shows ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(int))
```

```
  using assms PositiveSlopes_def Slopes_def PositiveIntegers_def
    int1.pos_slope_has_inv SlopeOp1_def SlopeOp2_def
    BoundedIntMaps_def SlopeEquivalenceRel_def
  by simp
```

The set of real numbers we are constructing is an ordered field.

```
theorem (in real1) reals_are_ord_field: shows
  IsAnOrdField(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
proof -
  let R = RealNumbers
  let A = RealAddition
  let M = RealMultiplication
  let r = OrderOnReals
  have ring1(R,A,M,r) and 0 ≠ 1
    using reals_are_ord_ring OrdRing_ZF_1_L2 real_zero_not_one
    by auto
  moreover have M {is commutative on} R
    using real_mult_commutative by simp
  moreover have
    ∀a∈PositiveSet(R,A,r). ∃b∈R. a·b = 1
  proof
    fix a assume a ∈ PositiveSet(R,A,r)
    then obtain f where I: f∈𝒮₊ and II: a = [f]
      using reals_are_ord_ring Real_ZF_1_2_L2
      by auto
    then have ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(int))
      using pos_slopes_have_inv by simp
    then obtain g where
      III: g∈𝒮 and IV: f∼g and V: ∃h∈𝒮. g∘h ∼ id(int)
      by auto
    from V obtain h where VII: h∈𝒮 and VIII: g∘h ∼ id(int)
      by auto
    from I III IV have [f] = [g]
      using Real_ZF_1_2_L1 Slopes_def Real_ZF_1_1_L5
      by auto
    with II III VII VIII have a·[h] = 1
      using Real_ZF_1_1_L4  Real_ZF_1_1_L5A real_one_cl_identity
      by simp
    with VII show ∃b∈R. a·b = 1 using Real_ZF_1_1_L3
      by auto
  qed
  ultimately show thesis using ring1.OrdField_ZF_1_L4
    by simp
qed
```

Reals form a field.

```
lemma reals_are_field:
  shows IsAfield(RealNumbers,RealAddition,RealMultiplication)
  using real1.reals_are_ord_field OrdField_ZF_1_L1A
```

```
  by simp
```

Theorem proven in `field0` and `field1` contexts are valid as applied to real numbers.

```
lemma field_cntxts_ok: shows
  field0(RealNumbers,RealAddition,RealMultiplication)
  field1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  using reals_are_field real1.reals_are_ord_field
    field_field0 OrdField_ZF_1_L2 by auto
```

If a is positive, then a^{-1} is also positive.

```
lemma (in real1) Real_ZF_1_3_L1: assumes a ∈ ℝ₊
  shows a⁻¹ ∈ ℝ₊   a⁻¹ ∈ ℝ
  using assms field_cntxts_ok field1.OrdField_ZF_1_L8 PositiveSet_def
  by auto
```

A technical fact about multiplying strict inequality by the inverse of one of the sides.

```
lemma (in real1) Real_ZF_1_3_L2:
  assumes a ∈ ℝ₊ and a⁻¹ < b
  shows 1 < b·a
  using assms field_cntxts_ok field1.OrdField_ZF_2_L2
  by simp
```

If a is smaller than b, then $(b-a)^{-1}$ is positive.

```
lemma (in real1) Real_ZF_1_3_L3: assumes a<b
  shows (b-a)⁻¹ ∈ ℝ₊
  using assms field_cntxts_ok field1.OrdField_ZF_1_L9
  by simp
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse.

```
lemma (in real1) Real_ZF_1_3_L4:
  assumes A1: a∈ℝ  b∈ℝ₊ and A2: a·b < c
  shows a < c·b⁻¹
  using assms field_cntxts_ok field1.OrdField_ZF_2_L6
  by simp
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with the product initially on the right hand side.

```
lemma (in real1) Real_ZF_1_3_L4A:
  assumes A1: b∈ℝ  c∈ℝ₊ and A2: a < b·c
  shows a·c⁻¹ < b
  using assms field_cntxts_ok field1.OrdField_ZF_2_L6A
  by simp
```

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with the product initially on the right hand side.

```
lemma (in real1) Real_ZF_1_3_L4B:
  assumes A1: b∈ℝ  c∈ℝ₊ and A2: a ≤ b·c
  shows a·c⁻¹ ≤ b
  using assms field_cntxts_ok field1.OrdField_ZF_2_L5A
  by simp
```

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with the product initially on the left hand side.

```
lemma (in real1) Real_ZF_1_3_L4C:
  assumes A1: a∈ℝ  b∈ℝ₊ and A2: a·b ≤ c
  shows a ≤ c·b⁻¹
  using assms field_cntxts_ok field1.OrdField_ZF_2_L5
  by simp
```

A technical lemma about solving a strict inequality with three real numbers and inverse of a difference.

```
lemma (in real1) Real_ZF_1_3_L5:
  assumes a<b and (b-a)⁻¹ < c
  shows 1 + a·c < b·c
  using assms field_cntxts_ok field1.OrdField_ZF_2_L9
  by simp
```

We can multiply an inequality by the inverse of a positive number.

```
lemma (in real1) Real_ZF_1_3_L6:
  assumes a≤b  and c∈ℝ₊ shows a·c⁻¹ ≤ b·c⁻¹
  using assms field_cntxts_ok field1.OrdField_ZF_2_L3
  by simp
```

We can multiply a strict inequality by a positive number or its inverse.

```
lemma (in real1) Real_ZF_1_3_L7:
  assumes a<b  and c∈ℝ₊ shows
  a·c < b·c
  c·a < c·b
  a·c⁻¹ < b·c⁻¹
  using assms field_cntxts_ok field1.OrdField_ZF_2_L4
  by auto
```

An identity with three real numbers, inverse and cancelling.

```
lemma (in real1) Real_ZF_1_3_L8: assumesa∈ℝ  b∈ℝ b≠0  c∈ℝ
  shows a·b·(c·b⁻¹) = a·c
  using assms field_cntxts_ok field0.Field_ZF_2_L6
  by simp
```

63.5 Completeness

This goal of this section is to show that the order on real numbers is complete, that is every subset of reals that is bounded above has a smallest upper bound.

If m is an integer, then m^R is a real number. Recall that in real1 context m^R denotes the class of the slope $n \mapsto m \cdot n$.

```
lemma (in real1) real_int_is_real: assumes m ∈ int
  shows m^R ∈ ℝ
  using assms int1.Int_ZF_2_5_L1 Real_ZF_1_1_L3 by simp
```

The negative of the real embedding of an integer is the embedding of the negative of the integer.

```
lemma (in real1) Real_ZF_1_4_L1: assumes m ∈ int
  shows (-m)^R = -(m^R)
  using assms int1.Int_ZF_2_5_L3 int1.Int_ZF_2_5_L1 Real_ZF_1_1_L4A
  by simp
```

The embedding of sum of integers is the sum of embeddings.

```
lemma (in real1) Real_ZF_1_4_L1A: assumes m ∈ int  k ∈ int
  shows m^R + k^R = ((m+k)^R)
  using assms int1.Int_ZF_2_5_L1 SlopeOp1_def int1.Int_ZF_2_5_L3A
    Real_ZF_1_1_L4 by simp
```

The embedding of a difference of integers is the difference of embeddings.

```
lemma (in real1) Real_ZF_1_4_L1B: assumes A1: m ∈ int  k ∈ int
  shows m^R - k^R = (m-k)^R
proof -
  from A1 have (-k) ∈ int using int0.Int_ZF_1_1_L4
    by simp
  with A1 have (m-k)^R = m^R + (-k)^R
    using Real_ZF_1_4_L1A by simp
  with A1 show m^R - k^R = (m-k)^R
    using Real_ZF_1_4_L1 by simp
qed
```

The embedding of the product of integers is the product of embeddings.

```
lemma (in real1) Real_ZF_1_4_L1C: assumes m ∈ int  k ∈ int
  shows m^R · k^R = (m·k)^R
  using assms int1.Int_ZF_2_5_L1 SlopeOp2_def int1.Int_ZF_2_5_L3B
    Real_ZF_1_1_L4 by simp
```

For any real numbers there is an integer whose real version is greater or equal.

```
lemma (in real1) Real_ZF_1_4_L2: assumes A1: a∈ℝ
  shows ∃m∈int. a ≤ m^R
proof -
  from A1 obtain f where I: f∈𝒮 and II: a = [f]
    using Real_ZF_1_1_L3A by auto
  then have ∃m∈int. ∃g∈𝒮.
    {⟨n,m·n⟩ . n ∈ int} ∼ g ∧ (f∼g ∨ (g + (-f)) ∈ 𝒮₊)
    using int1.Int_ZF_2_5_L2 Slopes_def SlopeOp1_def
```

```
      BoundedIntMaps_def SlopeEquivalenceRel_def
      PositiveIntegers_def PositiveSlopes_def
    by simp
  then obtain m g where III: m∈int and IV: g∈𝒮 and
   {⟨n,m·n⟩ . n ∈ int} ∼ g ∧ (f∼g ∨ (g + (-f)) ∈ 𝒮₊)
    by auto
  then have m^R = [g] and f ∼ g ∨ (g + (-f)) ∈ 𝒮₊
    using Real_ZF_1_1_L5A by auto
  with I II IV have a ≤ m^R using Real_ZF_1_2_L12
    by simp
  with III show ∃m∈int. a ≤ m^R by auto
qed
```

For any real numbers there is an integer whose real version (embedding) is less or equal.

```
lemma (in real1) Real_ZF_1_4_L3: assumes A1: a∈ℝ
  shows {m ∈ int. m^R ≤ a} ≠ 0
proof -
  from A1 have (-a) ∈ ℝ using Real_ZF_1_1_L8
    by simp
  then obtain m where I: m∈int and II: (-a) ≤ m^R
    using Real_ZF_1_4_L2 by auto
  let k = GroupInv(int,IntegerAddition)(m)
  from A1 I II have k ∈ int and k^R ≤ a
    using Real_ZF_1_2_L13 Real_ZF_1_4_L1 int0.Int_ZF_1_1_L4
    by auto
  then show thesis by auto
qed
```

Embeddings of two integers are equal only if the integers are equal.

```
lemma (in real1) Real_ZF_1_4_L4:
  assumes A1: m ∈ int  k ∈ int and A2: m^R = k^R
  shows m=k
proof -
  let r = {⟨n, IntegerMultiplication ⟨m, n⟩⟩ . n ∈ int}
  let s = {⟨n, IntegerMultiplication ⟨k, n⟩⟩ . n ∈ int}
  from A1 A2 have r ∼ s
    using int1.Int_ZF_2_5_L1 AlmostHoms_def Real_ZF_1_1_L5
    by simp
  with A1 have
    m ∈ int  k ∈ int
    ⟨r,s⟩ ∈ QuotientGroupRel(AlmostHoms(int, IntegerAddition),
    AlHomOp1(int, IntegerAddition),FinRangeFunctions(int, int))
    using SlopeEquivalenceRel_def Slopes_def SlopeOp1_def
      BoundedIntMaps_def by auto
  then show m=k by (rule int1.Int_ZF_2_5_L6)
qed
```

The embedding of integers preserves the order.

```
lemma (in real1) Real_ZF_1_4_L5: assumes A1: m≤k
  shows mR ≤ kR
proof -
  let r = {⟨n, m·n⟩ . n ∈ int}
  let s = {⟨n, k·n⟩ . n ∈ int}
  from A1 have r ∈ 𝒮  s ∈ 𝒮
    using int0.Int_ZF_2_L1A int1.Int_ZF_2_5_L1 by auto
  moreover from A1 have r ∼ s ∨ s + (-r)  ∈ 𝒮+
    using Slopes_def SlopeOp1_def BoundedIntMaps_def SlopeEquivalenceRel_def
      PositiveIntegers_def PositiveSlopes_def
      int1.Int_ZF_2_5_L4 by simp
  ultimately show mR ≤ kR using Real_ZF_1_2_L12
    by simp
qed
```

The embedding of integers preserves the strict order.

```
lemma (in real1) Real_ZF_1_4_L5A: assumes A1: m≤k  m≠k
  shows mR < kR
proof -
  from A1 have mR ≤ kR using Real_ZF_1_4_L5
    by simp
  moreover
  from A1 have T: m ∈ int  k ∈ int
    using int0.Int_ZF_2_L1A by auto
  with A1 have mR ≠ kR using Real_ZF_1_4_L4
    by auto
  ultimately show mR < kR by simp
qed
```

For any real number there is a positive integer whose real version is (strictly) greater. This is Lemma 14 i) in [2].

```
lemma (in real1) Arthan_Lemma14i: assumes A1: a∈ℝ
  shows ∃n∈ℤ+. a < nR
proof -
  from A1 obtain m where I: m∈int and II: a ≤ mR
    using Real_ZF_1_4_L2 by auto
  let n = GreaterOf(IntegerOrder,1Z,m) + 1Z
  from I have T: n ∈ℤ+ and m ≤ n  m≠n
    using int0.Int_ZF_1_5_L7B by auto
  then have III: mR < nR
    using Real_ZF_1_4_L5A by simp
  with II have a < nR by (rule real_strict_ord_transit)
  with T show thesis by auto
qed
```

If one embedding is less or equal than another, then the integers are also less or equal.

```
lemma (in real1) Real_ZF_1_4_L6:
```

```
  assumes A1: k ∈ int  m ∈ int and A2: m^R ≤ k^R
  shows m≤k
proof -
  { assume A3: ⟨m,k⟩ ∉ IntegerOrder
    with A1 have ⟨k,m⟩ ∈ IntegerOrder
      by (rule int0.Int_ZF_2_L19)
    then have k^R ≤ m^R using Real_ZF_1_4_L5
      by simp
    with A2 have m^R = k^R by (rule real_ord_antisym)
    with A1 have k = m using Real_ZF_1_4_L4
      by auto
    moreover from A1 A3 have k≠m by (rule int0.Int_ZF_2_L19)
    ultimately have False by simp
  } then show m≤k by auto
qed
```

The floor function is well defined and has expected properties.

```
lemma (in real1) Real_ZF_1_4_L7: assumes A1: a∈ℝ
  shows
  IsBoundedAbove({m ∈ int. m^R ≤ a},IntegerOrder)
  {m ∈ int. m^R ≤ a} ≠ 0
  ⌊a⌋ ∈ int
  ⌊a⌋^R ≤ a
proof -
  let A = {m ∈ int. m^R ≤ a}
  from A1 obtain K where I: K∈int and II: a ≤ (K^R)
    using Real_ZF_1_4_L2 by auto
  { fix n assume n ∈ A
    then have III: n ∈ int and IV: n^R ≤ a
      by auto
    from IV II have (n^R) ≤ (K^R)
      by (rule real_ord_transitive)
    with I III have n≤K using Real_ZF_1_4_L6
      by simp
  } then have ∀n∈A. ⟨n,K⟩ ∈ IntegerOrder
    by simp
  then show IsBoundedAbove(A,IntegerOrder)
    by (rule Order_ZF_3_L10)
  moreover from A1 show A ≠ 0 using Real_ZF_1_4_L3
    by simp
  ultimately have Maximum(IntegerOrder,A) ∈ A
    by (rule int0.int_bounded_above_has_max)
  then show ⌊a⌋ ∈ int   ⌊a⌋^R ≤ a by auto
qed
```

Every integer whose embedding is less or equal a real number a is less or equal than the floor of a.

```
lemma (in real1) Real_ZF_1_4_L8:
  assumes A1: m ∈ int and A2: m^R ≤ a
```

```
  shows m ≤ ⌊a⌋
proof -
  let A = {m ∈ int. m^R ≤ a}
  from A2 have IsBoundedAbove(A,IntegerOrder) and A≠0
    using Real_ZF_1_2_L15 Real_ZF_1_4_L7 by auto
  then have ∀x∈A. ⟨x,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule int0.int_bounded_above_has_max)
  with A1 A2 show m ≤ ⌊a⌋ by simp
qed
```

Integer zero and one embed as real zero and one.

```
lemma (in real1) int_0_1_are_real_zero_one:
  shows 0_Z^R = 0  1_Z^R = 1
  using int1.Int_ZF_2_5_L7 BoundedIntMaps_def
    real_one_cl_identity real_zero_cl_bounded_map
  by auto
```

Integer two embeds as the real two.

```
lemma (in real1) int_two_is_real_two: shows 2_Z^R = 2
proof -
  have 2_Z^R = 1_Z^R + 1_Z^R
    using int0.int_zero_one_are_int Real_ZF_1_4_L1A
    by simp
  also have ... = 2 using int_0_1_are_real_zero_one
    by simp
  finally show 2_Z^R = 2 by simp
qed
```

A positive integer embeds as a positive (hence nonnegative) real.

```
lemma (in real1) int_pos_is_real_pos: assumes A1: p∈ℤ+
  shows
  p^R ∈ ℝ
  0 ≤ p^R
  p^R ∈ ℝ+
proof -
  from A1 have I: p ∈ int  0_Z ≤ p  0_Z ≠ p
    using PositiveSet_def by auto
  then have p^R ∈ ℝ  0_Z^R ≤ p^R
    using real_int_is_real Real_ZF_1_4_L5 by auto
  then show p^R ∈ ℝ  0 ≤ p^R
    using int_0_1_are_real_zero_one by auto
  moreover have 0 ≠ p^R
  proof -
    { assume 0 = p^R
      with I have False using int_0_1_are_real_zero_one
 int0.int_zero_one_are_int Real_ZF_1_4_L4 by auto
    } then show 0 ≠ p^R by auto
  qed
  ultimately show p^R ∈ ℝ+ using PositiveSet_def
```

```
    by simp
qed
```

The ordered field of reals we are constructing is archimedean, i.e., if x, y are its elements with y positive, then there is a positive integer M such that x is smaller than $M^R y$. This is Lemma 14 ii) in [2].

```
lemma (in real1) Arthan_Lemma14ii: assumes A1: x∈ℝ  y ∈ ℝ₊
  shows ∃M∈ℤ₊. x < M^R·y
proof -
  from A1 have
    ∃C∈ℤ₊. x < C^R and ∃D∈ℤ₊. y⁻¹ < D^R
    using Real_ZF_1_3_L1 Arthan_Lemma14i by auto
  then obtain C D where
    I: C∈ℤ₊ and II: x < C^R and
    III: D∈ℤ₊ and IV: y⁻¹ < D^R
    by auto
  let M = C·D
  from I III have
    T: M ∈ ℤ₊  C^R ∈ ℝ  D^R ∈ ℝ
    using int0.pos_int_closed_mul_unfold PositiveSet_def real_int_is_real
    by auto
  with A1 I III have C^R·(D^R·y) = M^R·y
    using PositiveSet_def Real_ZF_1_L6A Real_ZF_1_4_L1C
    by simp
  moreover from A1 I II IV have
    x < C^R·(D^R·y)
    using int_pos_is_real_pos Real_ZF_1_3_L2 Real_ZF_1_2_L25
    by auto
  ultimately have x < M^R·y
    by auto
  with T show thesis by auto
qed
```

Taking the floor function preserves the order.

```
lemma (in real1) Real_ZF_1_4_L9: assumes A1: a≤b
  shows ⌊a⌋ ≤ ⌊b⌋
proof -
  from A1 have T: a∈ℝ using Real_ZF_1_2_L15
    by simp
  with A1 have ⌊a⌋^R ≤ a and a≤b
    using Real_ZF_1_4_L7 by auto
  then have ⌊a⌋^R ≤ b by (rule real_ord_transitive)
  moreover from T have ⌊a⌋ ∈ int using Real_ZF_1_4_L7
    by simp
 ultimately show ⌊a⌋ ≤ ⌊b⌋ using Real_ZF_1_4_L8
   by simp
qed
```

If S is bounded above and p is a positive interger, then $\Gamma(S, p)$ is well defined.

```
lemma (in real1) Real_ZF_1_4_L10:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and A2: p∈ℤ+
  shows
  IsBoundedAbove({⌊pR·x⌋. x∈S},IntegerOrder)
  Γ(S,p) ∈ {⌊pR·x⌋. x∈S}
  Γ(S,p) ∈ int
proof -
  let A = {⌊pR·x⌋. x∈S}
  from A1 obtain X where I: ∀x∈S. x≤X
    using IsBoundedAbove_def by auto
  { fix m assume m ∈ A
    then obtain x where x∈S and II: m = ⌊pR·x⌋
      by auto
    with I have x≤X by simp
    moreover from A2 have 0 ≤ pR using int_pos_is_real_pos
      by simp
    ultimately have pR·x ≤ pR·X using Real_ZF_1_2_L14
      by simp
    with II have m ≤ ⌊pR·X⌋ using Real_ZF_1_4_L9
      by simp
  } then have ∀m∈A. ⟨m,⌊pR·X⌋⟩ ∈ IntegerOrder
    by auto
  then show II: IsBoundedAbove(A,IntegerOrder)
    by (rule Order_ZF_3_L10)
  moreover from A1 have III: A ≠ 0 by simp
  ultimately have Maximum(IntegerOrder,A) ∈ A
    by (rule int0.int_bounded_above_has_max)
  moreover from II III have Maximum(IntegerOrder,A) ∈ int
    by (rule int0.int_bounded_above_has_max)
  ultimately show Γ(S,p) ∈ {⌊pR·x⌋. x∈S} and Γ(S,p) ∈ int
    by auto
qed
```

If p is a positive integer, then for all $s \in S$ the floor of $p \cdot x$ is not greater that $\Gamma(S, p)$.

```
lemma (in real1) Real_ZF_1_4_L11:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and A2: x∈S and A3: p∈ℤ+
  shows ⌊pR·x⌋ ≤ Γ(S,p)
proof -
  let A = {⌊pR·x⌋. x∈S}
  from A2 have S≠0 by auto
  with A1 A3 have IsBoundedAbove(A,IntegerOrder)  A ≠ 0
    using  Real_ZF_1_4_L10 by auto
  then have ∀x∈A. ⟨x,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule int0.int_bounded_above_has_max)
  with A2 show ⌊pR·x⌋ ≤ Γ(S,p) by simp
qed
```

The candidate for supremum is an integer mapping with values given by Γ.

```
lemma (in real1) Real_ZF_1_4_L12:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: g = {⟨p,Γ(S,p)⟩. p∈ℤ₊}
  shows
  g : ℤ₊→int
  ∀n∈ℤ₊. g(n) = Γ(S,n)
proof -
  from A1 have ∀n∈ℤ₊. Γ(S,n) ∈ int using Real_ZF_1_4_L10
    by simp
  with A2 show I: g : ℤ₊→int using ZF_fun_from_total by simp
  { fix n assume n∈ℤ₊
    with A2 I have g(n) = Γ(S,n) using ZF_fun_from_tot_val
      by simp
  } then show ∀n∈ℤ₊. g(n) = Γ(S,n) by simp
qed
```

Every integer is equal to the floor of its embedding.

```
lemma (in real1) Real_ZF_1_4_L14: assumes A1: m ∈ int
  shows ⌊m^R⌋ = m
proof -
  let A = {n ∈ int. n^R ≤ m^R }
  have antisym(IntegerOrder) using int0.Int_ZF_2_L4
    by simp
  moreover from A1 have m ∈ A
    using real_int_is_real real_ord_refl by auto
  moreover from A1 have ∀n ∈ A. ⟨n,m⟩ ∈ IntegerOrder
    using Real_ZF_1_4_L6 by auto
  ultimately show ⌊m^R⌋ = m using Order_ZF_4_L14
    by auto
qed
```

Floor of (real) zero is (integer) zero.

```
lemma (in real1) floor_01_is_zero_one: shows
  ⌊0⌋ = 0_Z    ⌊1⌋ = 1_Z
proof -
  have ⌊(0_Z)^R⌋ = 0_Z and ⌊(1_Z)^R⌋ = 1_Z
    using int0.int_zero_one_are_int Real_ZF_1_4_L14
    by auto
  then show ⌊0⌋ = 0_Z and  ⌊1⌋ = 1_Z
    using int_0_1_are_real_zero_one
    by auto
qed
```

Floor of (real) two is (integer) two.

```
lemma (in real1) floor_2_is_two: shows ⌊2⌋ = 2_Z
proof -
  have ⌊(2_Z)^R⌋ = 2_Z
```

```
    using int0.int_two_three_are_int Real_ZF_1_4_L14
    by simp
  then show ⌊2⌋ = 2_Z using int_two_is_real_two
    by simp
qed
```

Floor of a product of embeddings of integers is equal to the product of integers.

```
lemma (in real1) Real_ZF_1_4_L14A: assumes A1: m ∈ int  k ∈ int
  shows  ⌊m^R·k^R⌋ = m·k
proof -
  from A1 have T: m·k ∈ int
    using int0.Int_ZF_1_1_L5 by simp
  from A1 have ⌊m^R·k^R⌋ = ⌊(m·k)^R⌋ using Real_ZF_1_4_L1C
    by simp
  with T show ⌊m^R·k^R⌋ = m·k using Real_ZF_1_4_L14
    by simp
qed
```

Floor of the sum of a number and the embedding of an integer is the floor of the number plus the integer.

```
lemma (in real1) Real_ZF_1_4_L15: assumes A1: x∈ℝ and A2: p ∈ int
  shows ⌊x + p^R⌋ = ⌊x⌋ + p
proof -
  let A = {n ∈ int. n^R ≤ x + p^R}
  have antisym(IntegerOrder) using int0.Int_ZF_2_L4
    by simp
  moreover have ⌊x⌋ + p ∈ A
  proof -
    from A1 A2 have ⌊x⌋^R ≤ x and p^R ∈ ℝ
      using Real_ZF_1_4_L7 real_int_is_real by auto
    then have ⌊x⌋^R + p^R ≤ x + p^R
      using add_num_to_ineq by simp
    moreover from A1 A2 have (⌊x⌋ + p)^R = ⌊x⌋^R + p^R
      using Real_ZF_1_4_L7 Real_ZF_1_4_L1A by simp
    ultimately have (⌊x⌋ + p)^R ≤ x + p^R
      by simp
    moreover from A1 A2 have ⌊x⌋ + p ∈ int
      using Real_ZF_1_4_L7 int0.Int_ZF_1_1_L5 by simp
    ultimately show ⌊x⌋ + p ∈ A by auto
  qed
  moreover have ∀n∈A. n ≤ ⌊x⌋ + p
  proof
    fix n assume n∈A
    then have I: n ∈ int and n^R ≤ x + p^R
      by auto
    with A1 A2 have n^R - p^R ≤ x
      using real_int_is_real Real_ZF_1_2_L19
      by simp
```

```
    with A2 I have ⌊(n-p)^R⌋ ≤ ⌊x⌋
      using Real_ZF_1_4_L1B Real_ZF_1_4_L9
      by simp
    moreover
    from A2 I have n-p ∈ int
      using int0.Int_ZF_1_1_L5 by simp
    then have ⌊(n-p)^R⌋ = n-p
      using Real_ZF_1_4_L14 by simp
    ultimately have n-p ≤ ⌊x⌋
      by simp
    with A2 I show n ≤ ⌊x⌋ + p
      using int0.Int_ZF_2_L9C by simp
  qed
  ultimately show ⌊x + p^R⌋ = ⌊x⌋ + p
    using Order_ZF_4_L14 by auto
qed
```

Floor of the difference of a number and the embedding of an integer is the floor of the number minus the integer.

```
lemma (in real1) Real_ZF_1_4_L16: assumes A1: x∈ℝ and A2: p ∈ int
  shows ⌊x - p^R⌋ = ⌊x⌋ - p
proof -
  from A2 have ⌊x - p^R⌋ = ⌊x + (-p)^R⌋
     using Real_ZF_1_4_L1 by simp
  with A1 A2 show ⌊x - p^R⌋ = ⌊x⌋ - p
    using int0.Int_ZF_1_1_L4 Real_ZF_1_4_L15 by simp
qed
```

The floor of sum of embeddings is the sum of the integers.

```
lemma (in real1) Real_ZF_1_4_L17: assumes m ∈ int  n ∈ int
  shows ⌊(m^R) + n^R⌋ = m + n
  using assms real_int_is_real Real_ZF_1_4_L15 Real_ZF_1_4_L14
  by simp
```

A lemma about adding one to floor.

```
lemma (in real1) Real_ZF_1_4_L17A: assumes A1: a∈ℝ
  shows 1 + ⌊a⌋^R = (1_Z + ⌊a⌋)^R
proof -
  have 1 + ⌊a⌋^R = 1_Z^R + ⌊a⌋^R
    using int_0_1_are_real_zero_one by simp
  with A1 show 1 + ⌊a⌋^R = (1_Z + ⌊a⌋)^R
    using int0.int_zero_one_are_int Real_ZF_1_4_L7 Real_ZF_1_4_L1A
    by simp
qed
```

The difference between the a number and the embedding of its floor is (strictly) less than one.

```
lemma (in real1) Real_ZF_1_4_L17B: assumes A1: a∈ℝ
```

```
  shows
  a - ⌊a⌋^R < 1
  a < (1_Z + ⌊a⌋)^R
proof -
  from A1 have T1: ⌊a⌋ ∈ int  ⌊a⌋^R ∈ ℝ and
    T2: 1 ∈ ℝ  a -  ⌊a⌋^R ∈ ℝ
    using Real_ZF_1_4_L7 real_int_is_real Real_ZF_1_L6 Real_ZF_1_L4
    by auto
  { assume 1 ≤ a -  ⌊a⌋^R
    with A1 T1 have ⌊1_Z^R + ⌊a⌋^R⌋ ≤ ⌊a⌋
      using Real_ZF_1_2_L21 Real_ZF_1_4_L9 int_0_1_are_real_zero_one
      by simp
    with T1 have False
      using int0.int_zero_one_are_int Real_ZF_1_4_L17
      int0.Int_ZF_1_2_L3AA by simp
  } then have I: ¬(1 ≤ a - ⌊a⌋^R) by auto
  with T2 show II: a - ⌊a⌋^R < 1
    by (rule Real_ZF_1_2_L20)
   with A1 T1 I II have
    a < 1 + ⌊a⌋^R
    using Real_ZF_1_2_L26 by simp
  with A1 show a < (1_Z + ⌊a⌋)^R
    using Real_ZF_1_4_L17A by simp
qed
```

The next lemma corresponds to Lemma 14 iii) in [2]. It says that we can find a rational number between any two different real numbers.

```
lemma (in real1) Arthan_Lemma14iii: assumes A1: x<y
  shows ∃M∈int. ∃N∈ℤ+.  x·N^R < M^R ∧ M^R < y·N^R
proof -
  from A1 have (y-x)^-1 ∈ ℝ+ using Real_ZF_1_3_L3
    by simp
  then have
    ∃N∈ℤ+. (y-x)^-1 < N^R
    using Arthan_Lemma14i PositiveSet_def by simp
  then obtain N where I: N∈ℤ+ and II: (y-x)^-1 < N^R
    by auto
  let M = 1_Z + ⌊x·N^R⌋
  from A1 I have
    T1: x∈ℝ  N^R ∈ ℝ  N^R ∈ ℝ+  x·N^R ∈ ℝ
    using Real_ZF_1_2_L15 PositiveSet_def real_int_is_real
      Real_ZF_1_L6 int_pos_is_real_pos by auto
  then have T2: M ∈ int using
    int0.int_zero_one_are_int Real_ZF_1_4_L7 int0.Int_ZF_1_1_L5
    by simp
  from T1 have III: x·N^R < M^R
    using Real_ZF_1_4_L17B by simp
  from T1 have (1 + ⌊x·N^R⌋^R) ≤ (1 + x·N^R)
    using Real_ZF_1_4_L7  Real_ZF_1_L4 real_ord_transl_inv
```

```
    by simp
  with T1 have M^R ≤ (1 + x·N^R)
    using Real_ZF_1_4_L17A by simp
  moreover from A1 II have (1 + x·N^R) < y·N^R
    using Real_ZF_1_3_L5 by simp
  ultimately have M^R < y·N^R
    by (rule real_strict_ord_transit)
  with I T2 III show thesis by auto
qed
```

Some estimates for the homomorphism difference of the floor function.

```
lemma (in real1) Real_ZF_1_4_L18: assumes A1: x∈ℝ  y∈ℝ
  shows
  abs(⌊x+y⌋ - ⌊x⌋ - ⌊y⌋) ≤ 2_Z
proof -
  from A1 have T:
    ⌊x⌋^R ∈ ℝ  ⌊y⌋^R ∈ ℝ
    x+y - (⌊x⌋^R) ∈ ℝ
     using Real_ZF_1_4_L7 real_int_is_real Real_ZF_1_L6
     by auto
  from A1 have
    0 ≤ x - (⌊x⌋^R) + (y - (⌊y⌋^R))
    x -  (⌊x⌋^R) + (y - (⌊y⌋^R)) ≤ 2
    using Real_ZF_1_4_L7 Real_ZF_1_2_L16 Real_ZF_1_2_L17
      Real_ZF_1_4_L17B Real_ZF_1_2_L18 by auto
  moreover from A1 T have
    x - (⌊x⌋^R) + (y - (⌊y⌋^R)) = x+y - (⌊x⌋^R) - (⌊y⌋^R)
    using Real_ZF_1_L7A by simp
  ultimately have
    0 ≤ x+y - (⌊x⌋^R) - (⌊y⌋^R)
    x+y - (⌊x⌋^R) - (⌊y⌋^R) ≤ 2
    by auto
  then have
    ⌊0⌋ ≤ ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋
    ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋ ≤ ⌊2⌋
    using Real_ZF_1_4_L9 by auto
  then have
    0_Z  ≤ ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋
    ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋ ≤ 2_Z
    using floor_01_is_zero_one floor_2_is_two by auto
  moreover from A1 have
    ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋ = ⌊x+y⌋ - ⌊x⌋ - ⌊y⌋
    using Real_ZF_1_L6 Real_ZF_1_4_L7 real_int_is_real Real_ZF_1_4_L16
    by simp
  ultimately have
    0_Z ≤ ⌊x+y⌋ - ⌊x⌋ - ⌊y⌋
    ⌊x+y⌋ - ⌊x⌋ - ⌊y⌋ ≤ 2_Z
    by auto
  then show abs(⌊x+y⌋ - ⌊x⌋ - ⌊y⌋) ≤ 2_Z
```

```
    using int0.Int_ZF_2_L16 by simp
qed
```

Suppose $S \neq \emptyset$ is bounded above and $\Gamma(S,m) = \lfloor m^R \cdot x \rfloor$ for some positive integer m and $x \in S$. Then if $y \in S, x \leq y$ we also have $\Gamma(S,m) = \lfloor m^R \cdot y \rfloor$.

```
lemma (in real1) Real_ZF_1_4_L20:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: n∈ℤ₊  x∈S and
  A3: Γ(S,n) = ⌊nᴿ·x⌋ and
  A4: y∈S  x≤y
  shows Γ(S,n) = ⌊nᴿ·y⌋
proof -
  from A2 A4 have ⌊nᴿ·x⌋ ≤ ⌊(nᴿ)·y⌋
    using int_pos_is_real_pos Real_ZF_1_2_L14 Real_ZF_1_4_L9
    by simp
  with A3 have ⟨Γ(S,n),⌊(nᴿ)·y⌋⟩ ∈ IntegerOrder
    by simp
  moreover from A1 A2 A4 have ⟨⌊nᴿ·y⌋,Γ(S,n)⟩ ∈ IntegerOrder
    using Real_ZF_1_4_L11 by simp
  ultimately show Γ(S,n) = ⌊nᴿ·y⌋
    by (rule int0.Int_ZF_2_L3)
qed
```

The homomorphism difference of $n \mapsto \Gamma(S,n)$ is bounded by 2 on positive integers.

```
lemma (in real1) Real_ZF_1_4_L21:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: m∈ℤ₊  n∈ℤ₊
  shows abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) ≤  2_Z
proof -
  from A2 have T: m+n ∈ ℤ₊ using int0.pos_int_closed_add_unfolded
    by simp
  with A1 A2 have
    Γ(S,m) ∈ {⌊mᴿ·x⌋. x∈S} and
    Γ(S,n) ∈ {⌊nᴿ·x⌋. x∈S} and
    Γ(S,m+n) ∈ {⌊(m+n)ᴿ·x⌋. x∈S}
    using Real_ZF_1_4_L10 by auto
  then obtain a b c where I: a∈S  b∈S  c∈S
    and II:
    Γ(S,m) = ⌊mᴿ·a⌋
    Γ(S,n) = ⌊nᴿ·b⌋
    Γ(S,m+n) = ⌊(m+n)ᴿ·c⌋
    by auto
  let d = Maximum(OrderOnReals,{a,b,c})
  from A1 I have a∈ℝ  b∈ℝ  c∈ℝ
    using Real_ZF_1_2_L23 by auto
  then have IV:
    d ∈ {a,b,c}
    d ∈ ℝ
```

```
    a ≤ d
    b ≤ d
    c ≤ d
    using Real_ZF_1_2_L24 by auto
  with I have V: d ∈ S by auto
  from A1 T I II IV V have Γ(S,m+n) = ⌊(m+n)^R·d⌋
    using Real_ZF_1_4_L20 by blast
  also from A2 have ... = ⌊((m^R)+(n^R))·d⌋
    using Real_ZF_1_4_L1A PositiveSet_def by simp
  also from A2 IV have ... = ⌊(m^R)·d + (n^R)·d⌋
    using PositiveSet_def real_int_is_real Real_ZF_1_L7
    by simp
  finally have  Γ(S,m+n) =  ⌊(m^R)·d + (n^R)·d⌋
    by simp
  moreover from A1 A2 I II IV V have Γ(S,m) = ⌊m^R·d⌋
    using Real_ZF_1_4_L20 by blast
  moreover from A1 A2 I II IV V have  Γ(S,n) = ⌊n^R·d⌋
    using Real_ZF_1_4_L20 by blast
  moreover from A1 T I II IV V have Γ(S,m+n) = ⌊(m+n)^R·d⌋
    using Real_ZF_1_4_L20 by blast
  ultimately have abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) =
    abs(⌊(m^R)·d + (n^R)·d⌋ - ⌊m^R·d⌋ - ⌊n^R·d⌋)
    by simp
  with A2 IV show
    abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) ≤  2_Z
    using PositiveSet_def real_int_is_real Real_ZF_1_L6
      Real_ZF_1_4_L18 by simp
qed
```

The next lemma provides sufficient condition for an odd function to be an almost homomorphism. It says for odd functions we only need to check that the homomorphism difference (denoted δ in the `real1` context) is bounded on positive integers. This is really proven in `Int_ZF_2.thy`, but we restate it here for convenience. Recall from `Group_ZF_3.thy` that `OddExtension` of a function defined on the set of positive elements (of an ordered group) is the only odd function that is equal to the given one when restricted to positive elements.

```
lemma (in real1) Real_ZF_1_4_L21A:
  assumes A1: f:ℤ₊→int  ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  shows OddExtension(int,IntegerAddition,IntegerOrder,f) ∈ 𝒮
  using A1 int1.Int_ZF_2_1_L24 by auto
```

The candidate for (a representant of) the supremum of a nonempty bounded above set is a slope.

```
lemma (in real1) Real_ZF_1_4_L22:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: g = {⟨p,Γ(S,p)⟩. p∈ℤ₊}
  shows OddExtension(int,IntegerAddition,IntegerOrder,g) ∈ 𝒮
```

```
proof -
  from A1 A2 have g: ℤ₊→int by (rule Real_ZF_1_4_L12)
  moreover have ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ 2_Z
  proof -
    { fix m n assume A3: m∈ℤ₊  n∈ℤ₊
      then have m+n ∈ ℤ₊  m∈ℤ₊  n∈ℤ₊
 using int0.pos_int_closed_add_unfolded
 by auto
      moreover from A1 A2 have ∀n∈ℤ₊. g(n) = Γ(S,n)
 by (rule Real_ZF_1_4_L12)
      ultimately have δ(g,m,n) = Γ(S,m+n) - Γ(S,m) - Γ(S,n)
 by simp
      moreover from A1 A3 have
 abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) ≤  2_Z
 by (rule Real_ZF_1_4_L21)
      ultimately have abs(δ(g,m,n)) ≤ 2_Z
 by simp
    } then show ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ 2_Z
      by simp
  qed
  ultimately show thesis by (rule Real_ZF_1_4_L21A)
qed
```

A technical lemma used in the proof that all elements of S are less or equal than the candidate for supremum of S.

```
lemma (in real1) Real_ZF_1_4_L23:
  assumes A1: f ∈ 𝒮 and A2: N ∈ int  M ∈ int and
  A3: ∀n∈ℤ₊. M·n ≤ f(N·n)
  shows M^R ≤ [f]·(N^R)
proof -
  let M_S = {⟨n, M·n⟩ . n ∈ int}
  let N_S = {⟨n, N·n⟩ . n ∈ int}
  from A1 A2 have T: M_S ∈ 𝒮  N_S ∈ 𝒮  f∘N_S ∈ 𝒮
    using int1.Int_ZF_2_5_L1 int1.Int_ZF_2_1_L11 SlopeOp2_def
    by auto
  moreover from A1 A2 A3 have M_S ∼ f∘N_S ∨ f∘N_S + (-M_S) ∈ 𝒮₊
    using int1.Int_ZF_2_5_L8 SlopeOp2_def SlopeOp1_def Slopes_def
      BoundedIntMaps_def SlopeEquivalenceRel_def PositiveIntegers_def
      PositiveSlopes_def by simp
  ultimately have [M_S] ≤ [f∘N_S] using Real_ZF_1_2_L12
    by simp
  with A1 T show M^R ≤ [f]·(N^R) using Real_ZF_1_1_L4
    by simp
qed
```

A technical lemma aimed used in the proof the candidate for supremum of S is less or equal than any upper bound for S.

```
lemma (in real1) Real_ZF_1_4_L23A:
  assumes A1: f ∈ 𝒮 and A2: N ∈ int  M ∈ int and
```

```
  A3: ∀n∈ℤ₊. f(N·n) ≤  M·n
  shows [f]·(N^R) ≤ M^R
proof -
  let M_S = {⟨n, M·n⟩ . n ∈ int}
  let N_S = {⟨n, N·n⟩ . n ∈ int}
  from A1 A2 have T: M_S ∈ 𝒮  N_S ∈ 𝒮  f∘N_S ∈ 𝒮
    using int1.Int_ZF_2_5_L1 int1.Int_ZF_2_1_L11 SlopeOp2_def
    by auto
  moreover from A1 A2 A3 have
    f∘N_S ∼ M_S ∨  M_S + (-(f∘N_S)) ∈ 𝒮₊
    using int1.Int_ZF_2_5_L9 SlopeOp2_def SlopeOp1_def Slopes_def
      BoundedIntMaps_def SlopeEquivalenceRel_def PositiveIntegers_def
      PositiveSlopes_def by simp
  ultimately have [f∘N_S] ≤ [M_S] using Real_ZF_1_2_L12
    by simp
  with A1 T show  [f]·(N^R)≤ M^R using Real_ZF_1_1_L4
    by simp
qed
```

The essential condition to claim that the candidate for supremum of S is greater or equal than all elements of S.

```
lemma (in real1) Real_ZF_1_4_L24:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: x<y  y∈S  and
  A4: N ∈ ℤ₊  M ∈ int and
  A5: M^R < y·N^R and A6: p ∈ ℤ₊
  shows p·M ≤ Γ(S,p·N)
proof -
  from A2 A4 A6 have T1:
    N^R ∈ ℝ₊   y∈ℝ   p^R ∈ ℝ₊
    p·N ∈ ℤ₊   (p·N)^R ∈ ℝ₊
    using int_pos_is_real_pos Real_ZF_1_2_L15
    int0.pos_int_closed_mul_unfold by auto
  with A4 A6 have T2:
    p ∈ int   p^R ∈ ℝ   N^R ∈ ℝ  N^R ≠ 0   M^R ∈ ℝ
    using real_int_is_real PositiveSet_def by auto
  from T1 A5 have ⌊(p·N)^R·(M^R·(N^R)^-1)⌋ ≤ ⌊(p·N)^R·y⌋
    using Real_ZF_1_3_L4A Real_ZF_1_3_L7 Real_ZF_1_4_L9
    by simp
  moreover from A1 A2 T1 have ⌊(p·N)^R·y⌋ ≤ Γ(S,p·N)
    using Real_ZF_1_4_L11 by simp
  ultimately have I: ⌊(p·N)^R·(M^R·(N^R)^-1)⌋ ≤ Γ(S,p·N)
    by (rule int_order_transitive)
  from A4 A6 have (p·N)^R·(M^R·(N^R)^-1) = p^R·N^R·(M^R·(N^R)^-1)
    using PositiveSet_def Real_ZF_1_4_L1C by simp
  with A4 T2 have ⌊(p·N)^R·(M^R·(N^R)^-1)⌋ = p·M
    using Real_ZF_1_3_L8 Real_ZF_1_4_L14A by simp
  with I show p·M ≤ Γ(S,p·N) by simp
qed
```

An obvious fact about odd extension of a function $p \mapsto \Gamma(s,p)$ that is used a couple of times in proofs.

```
lemma (in real1) Real_ZF_1_4_L24A:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and A2: p ∈ ℤ₊
  and A3:
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ₊})
  shows h(p) = Γ(S,p)
proof -
  let g = {⟨p,Γ(S,p)⟩. p∈ℤ₊}
  from A1 have I: g : ℤ₊→int using  Real_ZF_1_4_L12
    by blast
  with A2 A3 show h(p) = Γ(S,p)
    using int0.Int_ZF_1_5_L11 ZF_fun_from_tot_val
    by simp
qed
```

The candidate for the supremum of S is not smaller than any element of S.

```
lemma (in real1) Real_ZF_1_4_L25:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: ¬HasAmaximum(OrderOnReals,S) and
  A3: x∈S and A4:
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ₊})
  shows x ≤ [h]
proof -
  from A1 A2 A3 have
    S ⊆ ℝ  ¬HasAmaximum(OrderOnReals,S)  x∈S
    using Real_ZF_1_2_L23 by auto
  then have ∃y∈S. x<y by (rule Real_ZF_1_2_L27)
  then obtain y where I: y∈S and  II: x<y
    by auto
  from II have
    ∃M∈int. ∃N∈ℤ₊.  x·N^R < M^R ∧ M^R < y·N^R
    using Arthan_Lemma14iii by simp
  then obtain M N where III: M ∈ int  N∈ℤ₊ and
    IV: x·N^R < M^R  M^R < y·N^R
    by auto
  from II III IV have V: x ≤ M^R·(N^R)^-1
    using int_pos_is_real_pos Real_ZF_1_2_L15 Real_ZF_1_3_L4
    by auto
  from A3 have VI: S≠0 by auto
  with A1 A4 have T1: h ∈ 𝒮 using Real_ZF_1_4_L22
    by simp
  moreover from III have N ∈ int  M ∈ int
    using PositiveSet_def by auto
  moreover have ∀n∈ℤ₊. M·n ≤ h(N·n)
  proof
    let g = {⟨p,Γ(S,p)⟩. p∈ℤ₊}
    fix n assume A5: n∈ℤ₊
    with III have T2: N·n ∈ ℤ₊
```

```
      using int0.pos_int_closed_mul_unfold by simp
    from III A5 have
      N·n = n·N  and n·M = M·n
      using PositiveSet_def int0.Int_ZF_1_1_L5 by auto
    moreover
    from A1 I II III IV A5 have
      IsBoundedAbove(S,OrderOnReals)
      x<y  y∈S
      N ∈ ℤ₊  M ∈ int
      M^R < y·N^R  n ∈ ℤ₊
      by auto
    then have n·M ≤ Γ(S,n·N) by (rule Real_ZF_1_4_L24)
    moreover from A1 A4 VI T2 have h(N·n) = Γ(S,N·n)
      using Real_ZF_1_4_L24A by simp
    ultimately show M·n ≤ h(N·n) by auto
  qed
  ultimately have M^R ≤ [h]·N^R using Real_ZF_1_4_L23
    by simp
  with III T1 have M^R·(N^R)^-1 ≤ [h]
    using int_pos_is_real_pos Real_ZF_1_1_L3 Real_ZF_1_3_L4B
    by simp
  with V show x ≤ [h] by (rule real_ord_transitive)
qed
```

The essential condition to claim that the candidate for supremum of S is less or equal than any upper bound of S.

```
lemma (in real1) Real_ZF_1_4_L26:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: x≤y  x∈S  and
  A4: N ∈ ℤ₊  M ∈ int and
  A5: y·N^R < M^R  and A6: p ∈ ℤ₊
  shows ⌊(N·p)^R·x⌋ ≤ M·p
proof -
  from A2 A4 A6 have T:
    p·N ∈ ℤ₊  p ∈ int  N ∈ int
    p^R ∈ ℝ₊ p^R ∈ ℝ  N^R ∈ ℝ  x ∈ ℝ  y ∈ ℝ
    using int0.pos_int_closed_mul_unfold PositiveSet_def
      real_int_is_real Real_ZF_1_2_L15 int_pos_is_real_pos
    by auto
  with A2 have (p·N)^R·x ≤ (p·N)^R·y
    using int_pos_is_real_pos Real_ZF_1_2_L14A
    by simp
  moreover from A4 T have I:
    (p·N)^R = p^R·N^R
    (p·M)^R = p^R·M^R
    using Real_ZF_1_4_L1C by auto
  ultimately have (p·N)^R·x ≤ p^R·N^R·y
    by simp
  moreover
```

```
  from A5 T I have p^R·(y·N^R) < (p·M)^R
    using Real_ZF_1_3_L7 by simp
  with T have p^R·N^R·y < (p·M)^R using Real_ZF_1_1_L9
    by simp
  ultimately have (p·N)^R·x < (p·M)^R
    by (rule real_strict_ord_transit)
  then have ⌊(p·N)^R·x⌋ ≤ ⌊(p·M)^R⌋
    using Real_ZF_1_4_L9 by simp
  moreover
  from A4 T have p·M ∈ int using int0.Int_ZF_1_1_L5
    by simp
  then have ⌊(p·M)^R⌋ = p·M using Real_ZF_1_4_L14
    by simp
   moreover from A4 A6 have p·N = N·p and p·M = M·p
    using PositiveSet_def int0.Int_ZF_1_1_L5 by auto
  ultimately show ⌊(N·p)^R·x⌋ ≤ M·p by simp
qed
```

A piece of the proof of the fact that the candidate for the supremum of S is not greater than any upper bound of S, done separately for clarity (of mind).

```
lemma (in real1) Real_ZF_1_4_L27:
  assumes IsBoundedAbove(S,OrderOnReals)  S≠0 and
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ+})
  and p ∈ ℤ+
  shows ∃x∈S. h(p) = ⌊p^R·x⌋
  using assms Real_ZF_1_4_L10 Real_ZF_1_4_L24A by auto
```

The candidate for the supremum of S is not greater than any upper bound of S.

```
lemma (in real1) Real_ZF_1_4_L28:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0
  and A2: ∀x∈S. x≤y and A3:
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ+})
  shows [h] ≤ y
proof -
  from A1 obtain a where a∈S by auto
  with A1 A2 A3 have T: y∈ℝ  h ∈ 𝒮  [h] ∈ ℝ
    using Real_ZF_1_2_L15 Real_ZF_1_4_L22 Real_ZF_1_1_L3
    by auto
  { assume ¬([h] ≤ y)
    with T have y < [h] using Real_ZF_1_2_L28
      by blast
    then have ∃M∈int. ∃N∈ℤ+.  y·N^R < M^R ∧ M^R < [h]·N^R
      using Arthan_Lemma14iii by simp
    then obtain M N where I: M∈int  N∈ℤ+ and
      II: y·N^R < M^R  M^R < [h]·N^R
      by auto
    from I have III: N^R ∈ ℝ+ using int_pos_is_real_pos
```

```
      by simp
    have ∀p∈ℤ₊. h(N·p) ≤  M·p
    proof
      fix p assume A4: p∈ℤ₊
      with A1 A3 I have ∃x∈S. h(N·p) = ⌊(N·p)^R·x⌋
 using int0.pos_int_closed_mul_unfold Real_ZF_1_4_L27
 by simp
      with A1 A2 I II A4 show h(N·p) ≤  M·p
 using Real_ZF_1_4_L26 by auto
    qed
    with T I have [h]·N^R ≤ M^R
      using PositiveSet_def Real_ZF_1_4_L23A
      by simp
    with T III have [h] ≤  M^R·(N^R)^-1
      using Real_ZF_1_3_L4C by simp
    moreover from T II III have M^R·(N^R)^-1 < [h]
      using Real_ZF_1_3_L4A by simp
    ultimately have False using Real_ZF_1_2_L29 by blast
  } then show [h] ≤ y by auto
qed
```

Now we can prove that every nonempty subset of reals that is bounded above has a supremum. Proof by considering two cases: when the set has a maximum and when it does not.

```
lemma (in real1) real_order_complete:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0
  shows HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals{a})
proof -
  { assume HasAmaximum(OrderOnReals,S)
    with A1 have HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals{a})
      using Real_ZF_1_2_L10 IsAnOrdGroup_def IsPartOrder_def
 Order_ZF_5_L6 by simp }
  moreover
  { assume A2: ¬HasAmaximum(OrderOnReals,S)
    let h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩.
p∈ℤ₊})
    let r = OrderOnReals
    from A1 have antisym(OrderOnReals)  S≠0
      using Real_ZF_1_2_L10 IsAnOrdGroup_def IsPartOrder_def by auto
    moreover from A1 A2 have ∀x∈S. ⟨x,[h]⟩ ∈ r
      using Real_ZF_1_4_L25 by simp
    moreover from A1 have ∀y. (∀x∈S. ⟨x,y⟩ ∈ r) ⟶ ⟨[h],y⟩ ∈ r
      using Real_ZF_1_4_L28 by simp
    ultimately have HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals{a})
      by (rule Order_ZF_5_L5) }
  ultimately show thesis by blast
qed
```

Finally, we are ready to formulate the main result: that the construction of real numbers from the additive group of integers results in a complete ordered field. This theorem completes the construction. It was fun.

```
theorem eudoxus_reals_are_reals: shows
  IsAmodelOfReals(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  using real1.reals_are_ord_field real1.real_order_complete
    IsComplete_def IsAmodelOfReals_def by simp

end
```

64 Topology - introduction

```
theory Topology_ZF imports ZF1 Finite_ZF Fol1

begin
```

This theory file provides basic definitions and properties of topology, open and closed sets, closure and boundary.

64.1 Basic definitions and properties

A typical textbook defines a topology on a set X as a collection T of subsets of X such that $X \in T$, $\emptyset \in T$ and T is closed with respect to arbitrary unions and intersection of two sets. One can notice here that since we always have $\bigcup T = X$, the set on which the topology is defined (the "carrier" of the topology) can always be constructed from the topology itself and is superfluous in the definition. Moreover, as Marnix Klooster pointed out to me, the fact that the empty set is open can also be proven from other axioms. Hence, we define a topology as a collection of sets that is closed under arbitrary unions and intersections of two sets, without any mention of the set on which the topology is defined. Recall that `Pow(T)` is the powerset of T, so that if $M \in$ `Pow(T)` then M is a subset of T. The sets that belong to a topology T will be sometimes called "open in" T or just "open" if the topology is clear from the context.

Topology is a collection of sets that is closed under arbitrary unions and intersections of two sets.

```
definition
  IsATopology (_ {is a topology} [90] 91) where
  T {is a topology} ≡ ( ∀M ∈ Pow(T). ⋃M ∈ T ) ∧
  ( ∀U∈T. ∀ V∈T. U∩V ∈ T)
```

We define interior of a set A as the union of all open sets contained in A. We use `Interior(A,T)` to denote the interior of A.

```
definition
  Interior(A,T) ≡ ⋃ {U∈T. U ⊆ A}
```

A set is closed if it is contained in the carrier of topology and its complement is open.

```
definition
  IsClosed (infixl {is closed in} 90) where
  D {is closed in} T ≡ (D ⊆ ⋃T ∧ (⋃T)\D ∈ T)
```

To prove various properties of closure we will often use the collection of closed sets that contain a given set A. Such collection does not have a separate name in informal math. We will call it `ClosedCovers(A,T)`.

```
definition
  ClosedCovers(A,T) ≡ {D ∈ Pow(⋃T). D {is closed in} T ∧ A⊆D}
```

The closure of a set A is defined as the intersection of the collection of closed sets that contain A.

```
definition
  Closure(A,T) ≡ ⋂ ClosedCovers(A,T)
```

We also define boundary of a set as the intersection of its closure with the closure of the complement (with respect to the carrier).

```
definition
  Boundary(A,T) ≡ Closure(A,T) ∩ Closure(⋃T - A,T)
```

A set K is compact if for every collection of open sets that covers K we can choose a finite one that still covers the set. Recall that `FinPow(M)` is the collection of finite subsets of M (finite powerset of M), defined in IsarMathLib's `Finite_ZF` theory.

```
definition
  IsCompact (infixl {is compact in} 90) where
  K {is compact in} T ≡ (K ⊆ ⋃T ∧
  (∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ FinPow(M). K ⊆ ⋃N)))
```

A basic example of a topology: the powerset of any set is a topology.

```
lemma Pow_is_top: shows Pow(X) {is a topology}
proof -
  have ∀A∈Pow(Pow(X)). ⋃A ∈ Pow(X) by fast
  moreover have ∀U∈Pow(X). ∀V∈Pow(X). U∩V ∈ Pow(X) by fast
  ultimately show Pow(X) {is a topology} using IsATopology_def
    by auto
qed
```

Empty set is open.

```
lemma empty_open:
  assumes T {is a topology} shows ∅ ∈ T
proof -
  have 0 ∈ Pow(T) by simp
  with assms have ⋃0 ∈ T using IsATopology_def by blast
```

```
  thus 0 ∈ T by simp
qed
```

The carrier is open.

```
lemma carr_open: assumes T {is a topology} shows (⋃T) ∈ T
  using assms IsATopology_def by auto
```

Union of a collection of open sets is open.

```
lemma union_open: assumes T {is a topology} and ∀A∈𝒜. A ∈ T
  shows (⋃𝒜) ∈ T using assms IsATopology_def by auto
```

Union of a indexed family of open sets is open.

```
lemma union_indexed_open: assumes A1: T {is a topology} and A2: ∀i∈I.
P(i) ∈ T
  shows (⋃i∈I. P(i)) ∈ T  using assms union_open by simp
```

The complement of an open set is closed.

```
lemma compl_open_closed: assumes U∈T shows ((⋃T)\U) {is closed in}
T
proof -
  let D = (⋃T)\U
  from assms have D ⊆ ⋃T and (⋃T)\D = U by auto
  with assms show thesis unfolding IsClosed_def by simp
qed
```

The intersection of any nonempty collection of topologies on a set X is a topology.

```
lemma Inter_tops_is_top:
  assumes A1: ℳ ≠ 0 and A2: ∀T∈ℳ. T {is a topology}
  shows (⋂ℳ) {is a topology}
proof -
  { fix A assume A∈Pow(⋂ℳ)
    with A1 have ∀T∈ℳ. A∈Pow(T) by auto
    with A1 A2 have ⋃A ∈ ⋂ℳ using IsATopology_def
      by auto
  } then have ∀A. A∈Pow(⋂ℳ) ⟶ ⋃A ∈ ⋂ℳ by simp
  hence ∀A∈Pow(⋂ℳ). ⋃A ∈ ⋂ℳ by auto
  moreover
  { fix U V assume U ∈ ⋂ℳ and V ∈ ⋂ℳ
    then have ∀T∈ℳ. U ∈ T ∧ V ∈ T by auto
    with A1 A2 have ∀T∈ℳ. U∩V ∈ T using IsATopology_def
      by simp
  } then have ∀ U ∈ ⋂ℳ. ∀ V ∈ ⋂ℳ. U∩V ∈ ⋂ℳ
    by auto
  ultimately show (⋂ℳ) {is a topology}
    using IsATopology_def by simp
qed
```

Singletons are compact. Interestingly we do not have to assume that T is a topology for this. Note singletons do not have to be closed, we need the the space to be T_1 for that (see `Topology_ZF_1`).

```
lemma singl_compact:
  assumes x∈⋃T shows {x} {is compact in} T
  using assms singleton_in_finpow unfolding IsCompact_def
  by auto
```

We will now introduce some notation. In Isar, this is done by definining a "locale". Locale is kind of a context that holds some assumptions and notation used in all theorems proven in it. In the locale (context) below called `topology0` we assume that T is a topology. The interior of the set A (with respect to the topology in the context) is denoted `int(A)`. The closure of a set $A \subseteq \bigcup T$ is denoted `cl(A)` and the boundary is `∂A`.

```
locale topology0 =
  fixes T
  assumes topSpaceAssum: T {is a topology}

  fixes int
  defines int_def [simp]: int(A) ≡ Interior(A,T)

  fixes cl
  defines cl_def [simp]: cl(A) ≡ Closure(A,T)

  fixes boundary (∂_ [91] 92)
  defines boundary_def [simp]: ∂A ≡ Boundary(A,T)
```

Intersection of a finite nonempty collection of open sets is open.

```
lemma (in topology0) fin_inter_open_open: assumes N≠0 N ∈ FinPow(T)
  shows ⋂N ∈ T
  using topSpaceAssum assms IsATopology_def inter_two_inter_fin
  by simp
```

Having a topology T and a set X we can define the induced topology as the one consisting of the intersections of X with sets from T. The notion of a collection restricted to a set is defined in ZF1.thy.

```
lemma (in topology0) Top_1_L4:
  shows (T {restricted to} X) {is a topology}
proof -
  let S = T {restricted to} X
  have ∀A∈Pow(S). ⋃A ∈ S
  proof
    fix A assume A1: A∈Pow(S)
    have ∀V∈A. ⋃ {U ∈ T. V = U∩X} ∈ T
    proof -
      { fix V
 let M = {U ∈ T. V = U∩X}
```

```
 have M ∈ Pow(T) by auto
 with topSpaceAssum have ⋃M ∈ T using IsATopology_def by simp
      } thus thesis by simp
    qed
    hence {⋃{U∈T. V = U∩X}.V∈ A} ⊆ T by auto
    with topSpaceAssum have (⋃V∈A. ⋃{U∈T. V = U∩X}) ∈ T
      using IsATopology_def by auto
    then have (⋃V∈A. ⋃{U∈T. V = U∩X})∩ X ∈ S
      using RestrictedTo_def by auto
    moreover
    from A1 have ∀V∈A. ∃U∈T. V = U∩X
      using RestrictedTo_def by auto
    hence (⋃V∈A. ⋃{U∈T. V = U∩X})∩X = ⋃A by blast
    ultimately show ⋃A ∈ S by simp
  qed
  moreover have  ∀U∈S. ∀V∈S. U∩V ∈ S
  proof -
    { fix U V assume U∈S  V∈S
      then obtain U₁ V₁ where
 U₁ ∈ T ∧ U = U₁∩X and V₁ ∈ T ∧ V = V₁∩X
 using RestrictedTo_def by auto
      with topSpaceAssum have U₁∩V₁ ∈ T and U∩V = (U₁∩V₁)∩X
 using IsATopology_def by auto
      then have  U∩V ∈ S using RestrictedTo_def by auto
    } thus ∀U∈S. ∀ V∈S. U∩V ∈ S
      by simp
  qed
  ultimately show S {is a topology} using IsATopology_def
    by simp
qed
```

64.2 Interior of a set

In this section we show basic properties of the interior of a set.

Interior of a set A is contained in A.

```
lemma (in topology0) Top_2_L1: shows int(A) ⊆ A
  using Interior_def by auto
```

Interior is open.

```
lemma (in topology0) Top_2_L2: shows int(A) ∈ T
proof -
  have {U∈T. U⊆A} ∈ Pow(T) by auto
  with topSpaceAssum show int(A) ∈ T
    using IsATopology_def Interior_def by auto
qed
```

A set is open iff it is equal to its interior.

```
lemma (in topology0) Top_2_L3: shows U∈T ⟷ int(U) = U
```

```
proof
  assume U∈T then show int(U) = U
    using Interior_def by auto
next assume A1: int(U) = U
  have int(U) ∈ T using Top_2_L2 by simp
  with A1 show U∈T by simp
qed
```

Interior of the interior is the interior.

```
lemma (in topology0) Top_2_L4: shows int(int(A)) = int(A)
proof -
  let U = int(A)
  from topSpaceAssum have U∈T using Top_2_L2 by simp
  then show int(int(A)) = int(A) using Top_2_L3 by simp
qed
```

Interior of a bigger set is bigger.

```
lemma (in topology0) interior_mono:
  assumes A1: A⊆B shows int(A) ⊆ int(B)
proof -
  from A1 have ∀ U∈T. (U⊆A ⟶ U⊆B) by auto
  then show int(A) ⊆ int(B) using Interior_def by auto
qed
```

An open subset of any set is a subset of the interior of that set.

```
lemma (in topology0) Top_2_L5: assumes U⊆A and U∈T
  shows U ⊆ int(A)
  using assms Interior_def by auto
```

If a point of a set has an open neighboorhood contained in the set, then the point belongs to the interior of the set.

```
lemma (in topology0) Top_2_L6:  assumes ∃U∈T. (x∈U ∧ U⊆A)
  shows x ∈ int(A)
  using assms Interior_def by auto
```

A set is open iff its every point has a an open neighbourhood contained in the set. We will formulate this statement as two lemmas (implication one way and the other way). The lemma below shows that if a set is open then every point has a an open neighbourhood contained in the set.

```
lemma (in topology0) open_open_neigh:
  assumes A1: V∈T
  shows ∀x∈V. ∃U∈T. (x∈U ∧ U⊆V)
proof -
  from A1 have ∀x∈V. V∈T ∧ x ∈ V ∧ V ⊆ V by simp
  thus thesis by auto
qed
```

If every point of a set has a an open neighbourhood contained in the set then the set is open.

```
lemma (in topology0) open_neigh_open:
  assumes A1: ∀x∈V. ∃U∈T. (x∈U ∧ U⊆V)
  shows V∈T
proof -
  from A1 have V = int(V) using Top_2_L1 Top_2_L6
    by blast
  then show V∈T using Top_2_L3 by simp
qed
```

The intersection of interiors is a equal to the interior of intersections.

```
lemma (in topology0) int_inter_int: shows int(A) ∩ int(B) = int(A∩B)
proof
  have int(A) ∩ int(B) ⊆ A∩B using Top_2_L1 by auto
  moreover have int(A) ∩ int(B) ∈ T using Top_2_L2 IsATopology_def topSpaceAssum

    by auto
  ultimately show int(A) ∩ int(B) ⊆ int(A∩B) using Top_2_L5 by simp
  have A∩B ⊆ A and A∩B ⊆ B by auto
  then have int(A∩B) ⊆ int(A) and int(A∩B) ⊆ int(B) using interior_mono
by auto
  thus int(A∩B) ⊆ int(A) ∩ int(B) by auto
qed
```

64.3 Closed sets, closure, boundary.

This section is devoted to closed sets and properties of the closure and boundary operators.

The carrier of the space is closed.

```
lemma (in topology0) Top_3_L1: shows (⋃T) {is closed in} T
proof -
  have ⋃T - ⋃T = 0 by auto
  with topSpaceAssum have ⋃T - ⋃T ∈ T using IsATopology_def by auto
  then show thesis using IsClosed_def by simp
qed
```

Empty set is closed.

```
lemma (in topology0) Top_3_L2: shows 0 {is closed in} T
  using topSpaceAssum  IsATopology_def IsClosed_def by simp
```

The collection of closed covers of a subset of the carrier of topology is never empty. This is good to know, as we want to intersect this collection to get the closure.

```
lemma (in topology0) Top_3_L3:
  assumes A1: A ⊆ ⋃T shows ClosedCovers(A,T) ≠ 0
proof -
  from A1 have ⋃T ∈ ClosedCovers(A,T) using ClosedCovers_def Top_3_L1
    by auto
```

```
  thus thesis by auto
qed
```

Intersection of a nonempty family of closed sets is closed.

```
lemma (in topology0) Top_3_L4: assumes A1: K≠0 and
  A2: ∀D∈K. D {is closed in} T
  shows (⋂K) {is closed in} T
proof -
  from A2 have I: ∀D∈K. (D ⊆ ⋃T ∧ (⋃T - D)∈ T)
    using IsClosed_def by simp
  then have {⋃T - D. D∈ K} ⊆ T by auto
  with topSpaceAssum have (⋃ {⋃T - D. D∈ K}) ∈ T
    using IsATopology_def by auto
  moreover from A1 have ⋃ {⋃T - D. D∈ K} = ⋃T - ⋂K by fast
  moreover from A1 I have ⋂K ⊆ ⋃T by blast
  ultimately show (⋂K) {is closed in} T using  IsClosed_def
    by simp
qed
```

The union and intersection of two closed sets are closed.

```
lemma (in topology0) Top_3_L5:
  assumes A1: D₁ {is closed in} T   D₂ {is closed in} T
  shows
  (D₁∩D₂) {is closed in} T
  (D₁∪D₂) {is closed in} T
proof -
  have {D₁,D₂} ≠ 0 by simp
  with A1 have (⋂ {D₁,D₂}) {is closed in} T using Top_3_L4
    by fast
  thus (D₁∩D₂) {is closed in} T by simp
  from topSpaceAssum A1 have (⋃T - D₁) ∩ (⋃T - D₂) ∈ T
    using IsClosed_def IsATopology_def by simp
  moreover have (⋃T - D₁) ∩ (⋃T - D₂) = ⋃T - (D₁ ∪ D₂)
    by auto
  moreover from A1 have D₁ ∪ D₂ ⊆ ⋃T using IsClosed_def
    by auto
  ultimately show (D₁∪D₂) {is closed in} T using IsClosed_def
    by simp
qed
```

Finite union of closed sets is closed. To understand the proof recall that D ∈Pow(⋃T) means that D is a subset of the carrier of the topology.

```
lemma (in topology0) fin_union_cl_is_cl:
  assumes
  A1: N ∈ FinPow({D∈Pow(⋃T). D {is closed in} T})
  shows (⋃N) {is closed in} T
proof -
  let C = {D∈Pow(⋃T). D {is closed in} T}
  have 0∈C using Top_3_L2 by simp
```

```
  moreover have ∀A∈C. ∀B∈C. A∪B ∈ C
    using Top_3_L5 by auto
  moreover note A1
  ultimately have ⋃N ∈ C by (rule union_two_union_fin)
  thus (⋃N) {is closed in} T by simp
qed
```

Closure of a set is closed, hence the complement of the closure is open.

```
lemma (in topology0) cl_is_closed: assumes A ⊆ ⋃T
  shows cl(A) {is closed in} T and ⋃T - cl(A) ∈ T
  using assms Top_3_L3 Top_3_L4 Closure_def ClosedCovers_def IsClosed_def
  by auto
```

Closure of a bigger sets is bigger.

```
lemma (in topology0) top_closure_mono:
  assumes A1: B ⊆ ⋃T  and A2:A⊆B
  shows cl(A) ⊆ cl(B)
proof -
  from A2 have ClosedCovers(B,T)⊆ ClosedCovers(A,T)
    using ClosedCovers_def by auto
  with A1 show thesis using Top_3_L3 Closure_def by auto
qed
```

Boundary of a set is closed.

```
lemma (in topology0) boundary_closed:
  assumes A1: A ⊆ ⋃T shows ∂A {is closed in} T
proof -
  from A1 have ⋃T - A ⊆ ⋃T by fast
  with A1 show ∂A {is closed in} T
    using cl_is_closed Top_3_L5 Boundary_def by auto
qed
```

A set is closed iff it is equal to its closure.

```
lemma (in topology0) Top_3_L8: assumes A1: A ⊆ ⋃T
  shows A {is closed in} T ⟷ cl(A) = A
proof
  assume A {is closed in} T
  with A1 show cl(A) = A
    using Closure_def ClosedCovers_def by auto
next assume cl(A) = A
  then have ⋃T - A = ⋃T - cl(A) by simp
  with A1 show A {is closed in} T using cl_is_closed IsClosed_def
    by simp
qed
```

Complement of an open set is closed.

```
lemma (in topology0) Top_3_L9: assumes A1: A∈T
  shows (⋃T - A) {is closed in} T
```

```
proof -
  from topSpaceAssum A1 have ⋃T - (⋃T - A) = A and ⋃T - A ⊆ ⋃T
    using IsATopology_def by auto
  with A1 show (⋃T - A) {is closed in} T using IsClosed_def by simp
qed
```

A set is contained in its closure.

```
lemma (in topology0) cl_contains_set: assumes A ⊆ ⋃T shows A ⊆ cl(A)
  using assms Top_3_L1 ClosedCovers_def Top_3_L3 Closure_def by auto
```

Closure of a subset of the carrier is a subset of the carrier and closure of the complement is the complement of the interior.

```
lemma (in topology0) Top_3_L11: assumes A1: A ⊆ ⋃T
  shows
  cl(A) ⊆ ⋃T
  cl(⋃T \ A) = ⋃T \ int(A)
proof -
  from A1 show cl(A) ⊆ ⋃T using Top_3_L1 Closure_def ClosedCovers_def
    by auto
  from A1 have ⋃T - A ⊆ ⋃T - int(A) using Top_2_L1
    by auto
  moreover have I: ⋃T - int(A) ⊆ ⋃T   ⋃T - A ⊆ ⋃T by auto
  ultimately have cl(⋃T - A) ⊆ cl(⋃T - int(A))
    using top_closure_mono by simp
  moreover
  from I have (⋃T - int(A)) {is closed in} T
    using Top_2_L2 Top_3_L9 by simp
  with I have cl((⋃T) - int(A)) = ⋃T - int(A)
    using Top_3_L8 by simp
  ultimately have cl(⋃T - A) ⊆ ⋃T - int(A) by simp
  moreover
  from I have ⋃T - A ⊆ cl(⋃T - A) using cl_contains_set by simp
  hence ⋃T - cl(⋃T - A) ⊆ A and ⋃T - A ⊆ ⋃T  by auto
  then have ⋃T - cl(⋃T - A) ⊆ int(A)
    using cl_is_closed IsClosed_def Top_2_L5 by simp
  hence ⋃T - int(A) ⊆  cl(⋃T - A) by auto
  ultimately show cl(⋃T - A) = ⋃T - int(A) by auto
qed
```

Boundary of a set is the closure of the set minus the interior of the set.

```
lemma (in topology0) Top_3_L12: assumes A1: A ⊆ ⋃T
  shows ∂A = cl(A) - int(A)
proof -
  from A1 have ∂A = cl(A) ∩ (⋃T - int(A))
    using Boundary_def Top_3_L11 by simp
  moreover from A1 have
    cl(A) ∩ (⋃T - int(A)) = cl(A) - int(A)
    using Top_3_L11 by blast
  ultimately show ∂A = cl(A) - int(A) by simp
```

qed

If a set A is contained in a closed set B, then the closure of A is contained in B.

```
lemma (in topology0) Top_3_L13:
  assumes A1: B {is closed in} T   A⊆B
  shows cl(A) ⊆ B
proof -
  from A1 have B ⊆ ⋃T using IsClosed_def by simp
  with A1 show cl(A) ⊆ B using ClosedCovers_def Closure_def by auto
qed
```

If a set is disjoint with an open set, then we can close it and it will still be disjoint.

```
lemma (in topology0) disj_open_cl_disj:
  assumes A1: A ⊆ ⋃T  V∈T and  A2: A∩V = 0
  shows cl(A) ∩ V = 0
proof -
  from assms have A ⊆ ⋃T - V by auto
  moreover from A1 have (⋃T - V) {is closed in} T using Top_3_L9 by
simp
  ultimately have cl(A) - (⋃T - V) = 0
    using Top_3_L13 by blast
  moreover from A1 have cl(A) ⊆ ⋃T using cl_is_closed IsClosed_def
by simp
  then have cl(A) -(⋃T - V) = cl(A) ∩ V by auto
  ultimately show thesis by simp
qed
```

A reformulation of `disj_open_cl_disj`: If a point belongs to the closure of a set, then we can find a point from the set in any open neighboorhood of the point.

```
lemma (in topology0) cl_inter_neigh:
  assumes A ⊆ ⋃T and U∈T and x ∈ cl(A) ∩ U
  shows A∩U ≠ 0 using assms disj_open_cl_disj by auto
```

A reverse of `cl_inter_neigh`: if every open neiboorhood of a point has a nonempty intersection with a set, then that point belongs to the closure of the set.

```
lemma (in topology0) inter_neigh_cl:
  assumes A1: A ⊆ ⋃T and A2: x∈⋃T and A3: ∀U∈T. x∈U ⟶ U∩A ≠ 0
  shows x ∈ cl(A)
proof -
  { assume x ∉ cl(A)
    with A1 obtain D where D {is closed in} T and A⊆D and x∉D
      using Top_3_L3 Closure_def ClosedCovers_def by auto
    let U = (⋃T) - D
```

```
    from A2 ‹D {is closed in} T› ‹x∉D› ‹A⊆D› have U∈T x∈U and U∩A =
0
      unfolding IsClosed_def by auto
    with A3 have False by auto
  } thus thesis by auto
qed

end
```

65 Topology 1

```
theory Topology_ZF_1 imports Topology_ZF

begin
```

In this theory file we study separation axioms and the notion of base and subbase. Using the products of open sets as a subbase we define a natural topology on a product of two topological spaces.

65.1 Separation axioms

Topological spaces can be classified according to certain properties called "separation axioms". In this section we define what it means that a topological space is T_0, T_1 or T_2.

A topology on X is T_0 if for every pair of distinct points of X there is an open set that contains only one of them.

```
definition
  isT0 (_ {is T0} [90] 91) where
  T {is T0} ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
  (∃U∈T. (x∈U ∧ y∉U) ∨ (y∈U ∧ x∉U)))
```

A topology is T_1 if for every such pair there exist an open set that contains the first point but not the second.

```
definition
  isT1 (_ {is T1} [90] 91) where
  T {is T1} ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
  (∃U∈T. (x∈U ∧ y∉U)))
```

T_1 topological spaces are exactly those in which all singletons are closed.

```
lemma (in topology0) t1_def_alt:
  shows T {is T1} ⟷ (∀x∈⋃T. {x} {is closed in} T)
proof
  let X = ⋃T
  assume T1: T {is T1}
  { fix x assume x∈X
    let U = X-{x}
```

```
    have U ∈ T
    proof -
      let W = ⋃y∈U.⋃{V∈T. y∈V ∧ x∉V}
      { fix y assume y∈U
        with topSpaceAssum have (⋃{V∈T. y∈V ∧ x∉V}) ∈ T
          unfolding IsATopology_def by blast
      } hence ∀y∈U. (⋃{V∈T. y∈V ∧ x∉V}) ∈ T by blast
      with topSpaceAssum have W∈T by (rule union_indexed_open)
      have U = W
      proof
        show W⊆U by auto
        { fix y assume y∈U
          hence y∈X and y≠x by auto
          with T1 ‹x∈X› have  y ∈ ⋃{V∈T. y∈V ∧ x∉V}
            unfolding isT1_def by blast
          hence y∈W by blast
        } thus U ⊆ W by blast
      qed
      with ‹W∈T› show U∈T by simp
    qed
    with ‹x∈X› have (X-U) {is closed in} T and X\U = {x}
      using Top_3_L9 by auto
    hence {x} {is closed in} T by simp
  } thus ∀x∈X. {x} {is closed in} T by blast
next
  let X = ⋃T
  assume scl: ∀x∈⋃T. {x} {is closed in} T
  { fix x y assume x∈X y∈X x≠y
    let U = X-{y}
    from scl ‹x∈X› ‹y∈X› ‹x≠y› have U ∈ T x∈U ∧ y∉U
      unfolding IsClosed_def by auto
    then have ∃U∈T. (x∈U ∧ y∉U) by (rule witness_exists)
  } then show T {is T₁} unfolding isT1_def by blast
qed
```

A topology is T_2 (Hausdorff) if for every pair of points there exist a pair of disjoint open sets each containing one of the points. This is an important class of topological spaces. In particular, metric spaces are Hausdorff.

```
definition
  isT2 (_ {is T₂} [90] 91) where
  T {is T₂} ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
  (∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0))
```

A topology is regular if every closed set can be separated from a point in its complement by (disjoint) opens sets.

```
definition
  IsRegular (_ {is regular} 90)
  where T {is regular} ≡ ∀D. D {is closed in} T ⟶ (∀x∈⋃T-D.∃U∈T.∃V∈T.
D⊆U∧x∈V∧U∩V=0)
```

Some sources (e.g. Metamath) use a different definition of regularity: any open neighborhood has a closed subneighborhood. The next lemma shows the equivalence of this with our definition.

```
lemma is_regular_def_alt: assumes T {is a topology}
  shows T {is regular} ⟷ (∀W∈T. ∀x∈W. ∃V∈T. x∈V ∧ Closure(V,T)⊆W)
proof
  let X = ⋃T
  from assms(1) have cntx: topology0(T)
    unfolding topology0_def by simp
  assume T {is regular}
  { fix W x assume W∈T x∈W
    have ∃V∈T. x∈V ∧ Closure(V,T)⊆W
    proof -
      let D = X-W
      from cntx ‹W∈T› ‹T {is regular}› ‹x∈W›
      have ∃U∈T.∃V∈T. D⊆U∧x∈V∧U∩V=0
        using topology0.Top_3_L9 unfolding IsRegular_def by auto
      then obtain U V where U∈T V∈T D⊆U x∈V V∩U=0
        by blast
      from cntx ‹V∈T› have Closure(V,T) ⊆ X
        using topology0.Top_3_L11(1) by blast
      from cntx ‹V∈T› ‹U∈T› ‹V∩U=0› ‹D⊆U›
      have Closure(V,T) ∩ D = 0
        using topology0.disj_open_cl_disj by blast
      with ‹Closure(V,T) ⊆ X› ‹V∈T› ‹x∈V› show thesis
        by blast
    qed
  } thus ∀W∈T. ∀x∈W. ∃V∈T. x∈V ∧ Closure(V,T)⊆W
    by simp
next
  let X = ⋃T
  from assms(1) have cntx: topology0(T)
    unfolding topology0_def by simp
  assume regAlt: ∀W∈T. ∀x∈W. ∃V∈T. x∈V ∧ Closure(V,T)⊆W
  { fix A assume A {is closed in} T
    have ∀x∈X-A.∃U∈T.∃V∈T. A⊆U ∧ x∈V ∧ U∩V=0
    proof -
      { let W = X-A
        from ‹A {is closed in} T› have A⊆X and W∈T
          unfolding IsClosed_def by auto
        fix x assume x∈W
        with regAlt ‹W∈T› have ∃V∈T. x∈V ∧ Closure(V,T)⊆W
          by simp
        then obtain V where V∈T x∈V Closure(V,T)⊆W
          by auto
        let U = X-Closure(V,T)
        from cntx ‹V∈T› have V⊆X and V⊆Closure(V,T)
          using topology0.cl_contains_set by auto
        with cntx ‹A⊆X› ‹Closure(V,T)⊆W›
```

```
      have U∈T A⊆U U∩V = 0
        using topology0.cl_is_closed(2) by auto
      with ‹V∈T› ‹x∈V› have ∃U∈T.∃V∈T. A⊆U ∧ x∈V ∧ U∩V=0
        by blast
    } thus thesis by blast
  qed
} then show T {is regular} unfolding IsRegular_def
  by blast
qed
```

If a topology is T_1 then it is T_0. We don't really assume here that T is a topology on X. Instead, we prove the relation between isT0 condition and isT1.

```
lemma T1_is_T0: assumes A1: T {is T₁} shows T {is T₀}
proof -
  from A1 have ∀ x y. x ∈ ⋃T ∧ y ∈ ⋃T ∧ x≠y ⟶
    (∃U∈T. x∈U ∧ y∉U)
    using isT1_def by simp
  then have ∀ x y. x ∈ ⋃T ∧ y ∈ ⋃T ∧ x≠y ⟶
    (∃U∈T. x∈U ∧ y∉U ∨ y∈U ∧ x∉U)
    by auto
  then show T {is T₀} using isT0_def by simp
qed
```

If a topology is T_2 then it is T_1.

```
lemma T2_is_T1: assumes A1: T {is T₂} shows T {is T₁}
proof -
  { fix x y assume x ∈ ⋃T  y ∈ ⋃T  x≠y
    with A1 have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0
      using isT2_def by auto
    then have ∃U∈T. x∈U ∧ y∉U by auto
  } then have ∀ x y. x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y ⟶
      (∃U∈T. x∈U ∧ y∉U) by simp
  then show T {is T₁} using isT1_def by simp
qed
```

In a T_0 space two points that can not be separated by an open set are equal. Proof by contradiction.

```
lemma Top_1_1_L1: assumes A1: T {is T₀} and A2: x ∈ ⋃T  y ∈ ⋃T
  and A3: ∀U∈T. (x∈U ⟷ y∈U)
  shows x=y
proof -
  { assume x≠y
    with A1 A2 have ∃U∈T. x∈U ∧ y∉U ∨ y∈U ∧ x∉U
      using isT0_def by simp
    with A3 have False by auto
  } then show x=y by auto
qed
```

A topology is T_3 if it is regular and T_0. T_3 spaces are called "regular Hausdorff", which is a bit confusing as the definition requires the space to be T_0 rather than T_2. It is ok though as we we will show that T_3 as defined here implies T_2 so indeed T_3 spaces are regular and Hausdorff. In some older sources the definitions of a regular and a T_3 space are swapped. We follow the terminology from Wikipedia's "Separation axiom" entry, where T_3 implies "regular".

```
definition
  isT3 (_ {is T₃} [90] 91) where
  T {is T₃} ≡ (T {is regular}) ∧ T {is T₀}
```

If a topology is T_3 then it is T_2. It's interesting that even here we do not have to assume that T is a topology.

```
lemma T3_is_T2: assumes T {is T₃} shows T {is T₂}
proof -
  let X = ⋃T
  { fix x y assume x∈X y∈X x≠y
    with assms obtain U where U∈T and (x∈U ∧ y∉U) ∨ (y∈U ∧ x∉U)
      unfolding isT3_def isT0_def by auto
    let F = X\U
    from assms ‹U∈T› have I: ∀t∈X\F.∃V∈T.∃W∈T. F⊆V ∧ t∈W ∧ V∩W=∅
      unfolding isT3_def IsRegular_def using compl_open_closed by blast
    note ‹(x∈U ∧ y∉U) ∨ (y∈U ∧ x∉U)›
    moreover
    { assume y∈U and x∉U
      with ‹x∈X› have x∈F by simp
      from I ‹y∈U› ‹U∈T› have ∃V∈T.∃W∈T. F⊆V ∧ y∈W ∧ V∩W=∅ by auto
      with ‹x∈F› have ∃V∈T.∃W∈T. x∈V ∧ y∈W ∧ V∩W=∅ by blast
    }
    moreover
    { assume x∈U and y∉U
      with ‹y∈X› have y∈F by simp
      from I ‹U∈T› ‹x∈U› have ∃V∈T.∃W∈T. F⊆V ∧ x∈W ∧ V∩W=∅ by auto
      with ‹y∈F› have ∃V∈T.∃W∈T. y∈V ∧ x∈W ∧ V∩W=∅ by blast
    }
    ultimately have ∃V∈T.∃W∈T. x∈V ∧ y∈W ∧ V∩W=∅ by blast
  } then show T {is T₂} unfolding isT2_def by simp
qed
```

Sometimes T_3 space is defined as one that is regular and T_1 (rather than T_0). The next lemma shows that this definition is equivalent to the standard one.

```
lemma T3_def_alt: shows T {is T₃} ⟷ (T {is regular}) ∧ T {is T₁}
  using T3_is_T2 T2_is_T1 T1_is_T0 unfolding isT3_def by auto
```

65.2 Bases and subbases

Sometimes it is convenient to talk about topologies in terms of their bases and subbases. These are certain collections of open sets that define the whole topology.

A base of topology is a collection of open sets such that every open set is a union of the sets from the base.

```
definition
  IsAbaseFor (infixl {is a base for} 65) where
  B {is a base for} T ≡ B⊆T ∧ T = {⋃A. A∈Pow(B)}
```

A subbase is a collection of open sets such that finite intersection of those sets form a base.

```
definition
  IsAsubBaseFor (infixl {is a subbase for} 65) where
  B {is a subbase for} T ≡
  B ⊆ T ∧ {⋂A. A ∈ FinPow(B)} {is a base for} T
```

Below we formulate a condition that we will prove to be necessary and sufficient for a collection B of open sets to form a base. It says that for any two sets U, V from the collection B we can find a point $x \in U \cap V$ with a neighboorhod from B contained in $U \cap V$.

```
definition
  SatisfiesBaseCondition (_ {satisfies the base condition} [50] 50)
  where
  B {satisfies the base condition} ≡
  ∀U V. ((U∈B ∧ V∈B) ⟶ (∀x ∈ U∩V. ∃W∈B. x∈W ∧ W ⊆ U∩V))
```

A collection that is closed with respect to intersection satisfies the base condition.

```
lemma inter_closed_base: assumes ∀U∈B.(∀V∈B. U∩V ∈ B)
  shows  B {satisfies the base condition}
proof -
    { fix U V x assume U∈B and V∈B and x ∈ U∩V
      with assms have ∃W∈B. x∈W ∧ W ⊆ U∩V by blast
    } then show thesis using SatisfiesBaseCondition_def by simp
qed
```

Each open set is a union of some sets from the base.

```
lemma Top_1_2_L1: assumes B {is a base for} T  and U∈T
  shows ∃A∈Pow(B). U = ⋃A
  using assms IsAbaseFor_def by simp
```

Elements of base are open.

```
lemma base_sets_open:
  assumes B {is a base for} T and U ∈ B
```

```
  shows U ∈ T
  using assms IsAbaseFor_def by auto
```

A base defines topology uniquely.

```
lemma same_base_same_top:
  assumes B {is a base for} T and B {is a base for} S
  shows T = S
  using assms IsAbaseFor_def by simp
```

Every point from an open set has a neighboorhood from the base that is contained in the set.

```
lemma point_open_base_neigh:
  assumes A1: B {is a base for} T and A2: U∈T and A3: x∈U
  shows ∃V∈B. V⊆U ∧ x∈V
proof -
  from A1 A2 obtain A where A ∈ Pow(B) and U = ⋃A
    using Top_1_2_L1 by blast
  with A3 obtain V where V∈A and x∈V by auto
  with ‹A ∈ Pow(B)› ‹U = ⋃A› show thesis by auto
qed
```

A criterion for a collection to be a base for a topology that is a slight reformulation of the definition. The only thing different that in the definition is that we assume only that every open set is a union of some sets from the base. The definition requires also the opposite inclusion that every union of the sets from the base is open, but that we can prove if we assume that T is a topology.

```
lemma is_a_base_criterion: assumes A1: T {is a topology}
  and A2: B ⊆ T and A3: ∀V ∈ T. ∃A ∈ Pow(B). V = ⋃A
  shows B {is a base for} T
proof -
  from A3 have T ⊆ {⋃A. A∈Pow(B)} by auto
  moreover have {⋃A. A∈Pow(B)} ⊆ T
  proof
    fix U assume U ∈ {⋃A. A∈Pow(B)}
    then obtain A where A ∈ Pow(B) and U = ⋃A
      by auto
    with ‹B ⊆ T› have A ∈ Pow(T) by auto
    with A1 ‹U = ⋃A› show U ∈ T
      unfolding IsATopology_def by simp
  qed
  ultimately have T = {⋃A. A∈Pow(B)} by auto
  with A2 show B {is a base for} T
    unfolding IsAbaseFor_def by simp
qed
```

A necessary condition for a collection of sets to be a base for some topology : every point in the intersection of two sets in the base has a neighboorhood from the base contained in the intersection.

```
lemma Top_1_2_L2:
  assumes A1:∃T. T {is a topology} ∧ B {is a base for} T
  and A2: V∈B  W∈B
  shows ∀ x ∈ V∩W. ∃U∈B. x∈U ∧ U ⊆ V ∩ W
proof -
  from A1 obtain T where
    D1: T {is a topology}   B {is a base for} T
    by auto
  then have B ⊆ T using IsAbaseFor_def by auto
  with A2 have V∈T and W∈T using IsAbaseFor_def by auto
  with D1 have ∃A∈Pow(B). V∩W = ⋃A using IsATopology_def Top_1_2_L1
    by auto
  then obtain A where A ⊆ B and V ∩ W = ⋃A by auto
  then show ∀ x ∈ V∩W. ∃U∈B. (x∈U ∧ U ⊆ V ∩ W) by auto
qed
```

We will construct a topology as the collection of unions of (would-be) base. First we prove that if the collection of sets satisfies the condition we want to show to be sufficient, the the intersection belongs to what we will define as topology (am I clear here?). Having this fact ready simplifies the proof of the next lemma. There is not much topology here, just some set theory.

```
lemma Top_1_2_L3:
  assumes A1: ∀x∈ V∩W . ∃U∈B. x∈U ∧ U ⊆ V∩W
  shows V∩W ∈ {⋃A. A∈Pow(B)}
proof
  let A = ⋃x∈V∩W. {U∈B. x∈U ∧ U ⊆ V∩W}
  show A∈Pow(B) by auto
  from A1 show V∩W = ⋃A by blast
qed
```

The next lemma is needed when proving that the would-be topology is closed with respect to taking intersections. We show here that intersection of two sets from this (would-be) topology can be written as union of sets from the topology.

```
lemma Top_1_2_L4:
  assumes A1:  U₁ ∈ {⋃A. A∈Pow(B)}   U₂ ∈ {⋃A. A∈Pow(B)}
  and A2: B {satisfies the base condition}
  shows ∃C. C ⊆ {⋃A. A∈Pow(B)} ∧ U₁∩U₂ = ⋃C
proof -
  from A1 A2 obtain A₁ A₂ where
    D1: A₁∈ Pow(B)  U₁ = ⋃A₁  A₂ ∈ Pow(B)  U₂ = ⋃A₂
    by auto
  let C = ⋃U∈A₁.{U∩V. V∈A₂}
  from D1 have (∀U∈A₁. U∈B) ∧ (∀V∈A₂. V∈B) by auto
  with A2 have C ⊆ {⋃A . A ∈ Pow(B)}
    using Top_1_2_L3 SatisfiesBaseCondition_def by auto
  moreover from D1 have U₁ ∩ U₂ = ⋃C by auto
  ultimately show thesis by auto
```

```
qed
```

If B satisfies the base condition, then the collection of unions of sets from B is a topology and B is a base for this topology.

```
theorem Top_1_2_T1:
  assumes A1: B {satisfies the base condition}
  and A2: T = {⋃A. A∈Pow(B)}
  shows T {is a topology} and B {is a base for} T
proof -
  show T {is a topology}
  proof -
    have I: ∀C∈Pow(T). ⋃C ∈ T
    proof -
      { fix C assume A3: C ∈ Pow(T)
        let Q = ⋃ {⋃{A∈Pow(B). U = ⋃A}. U∈C}
        from A2 A3 have ∀U∈C. ∃A∈Pow(B). U = ⋃A by auto
        then have ⋃Q = ⋃C using ZF1_1_L10 by simp
        moreover from A2 have ⋃Q ∈ T by auto
        ultimately have ⋃C ∈ T by simp
      } thus ∀C∈Pow(T). ⋃C ∈ T by auto
    qed
    moreover have ∀U∈T. ∀ V∈T. U∩V ∈ T
    proof -
      { fix U V assume  U ∈ T  V ∈ T
        with A1 A2 have ∃C.(C ⊆ T ∧ U∩V = ⋃C)
        using Top_1_2_L4 by simp
        then obtain C where C ⊆ T and  U∩V = ⋃C
          by auto
          with I have U∩V ∈ T by simp
      } then show ∀U∈T. ∀ V∈T. U∩V ∈ T by simp
    qed
    ultimately show T {is a topology} using IsATopology_def
      by simp
  qed
  from A2 have B⊆T by auto
  with A2 show B {is a base for} T using IsAbaseFor_def
    by simp
qed
```

The carrier of the base and topology are the same.

```
lemma Top_1_2_L5: assumes B {is a base for} T
  shows ⋃T = ⋃B
  using assms IsAbaseFor_def by auto
```

If B is a base for T, then T is the smallest topology containing B.

```
lemma base_smallest_top:
  assumes A1: B {is a base for} T and  A2: S {is a topology} and A3:
B⊆S
  shows T⊆S
```

```
proof
  fix U assume U∈T
  with A1 obtain B_U where B_U ⊆ B and U = ⋃B_U using IsAbaseFor_def
by auto
  with A3 have B_U ⊆ S by auto
  with A2 ‹U = ⋃B_U› show U∈S using IsATopology_def by simp
qed
```

If B is a base for T and B is a topology, then $B = T$.

```
lemma base_topology: assumes B {is a topology} and B {is a base for}
T
  shows B=T using assms base_sets_open base_smallest_top by blast
```

65.3 Product topology

In this section we consider a topology defined on a product of two sets.

Given two topological spaces we can define a topology on the product of the carriers such that the cartesian products of the sets of the topologies are a base for the product topology. Recall that for two collections S, T of sets the product collection is defined (in ZF1.thy) as the collections of cartesian products $A \times B$, where $A \in S, B \in T$. The $T \times_t S$ notation is defined as an alternative to the verbose ProductTopology(T,S)).

```
definition ProductTopology (infixl ×_t 65) where
  T ×_t S  ≡ {⋃W. W ∈ Pow(ProductCollection(T,S))}
```

The product collection satisfies the base condition.

```
lemma Top_1_4_L1:
  assumes A1: T {is a topology}   S {is a topology}
  and A2: A ∈ ProductCollection(T,S)  B ∈ ProductCollection(T,S)
  shows ∀x∈(A∩B). ∃W∈ProductCollection(T,S). (x∈W ∧ W ⊆ A ∩ B)
proof
  fix x assume A3: x ∈ A∩B
  from A2 obtain U1 V1 U2 V2 where
    D1: U1∈T  V1∈S   A=U1×V1  U2∈T  V2∈S   B=U2×V2
    using ProductCollection_def by auto
  let W = (U1∩U2) × (V1∩V2)
  from A1 D1 have U1∩U2 ∈ T and V1∩V2 ∈ S
    using IsATopology_def by auto
  then have W ∈ ProductCollection(T,S) using ProductCollection_def
    by auto
  moreover from A3 D1 have x∈W and W ⊆ A∩B by auto
  ultimately have ∃W. (W ∈ ProductCollection(T,S) ∧ x∈W ∧ W ⊆ A∩B)
    by auto
  thus ∃W∈ProductCollection(T,S). (x∈W ∧ W ⊆ A ∩ B) by auto
qed
```

The product topology is indeed a topology on the product.

```
theorem Top_1_4_T1: assumes A1: T {is a topology}  S {is a topology}
  shows
  (T×tS) {is a topology}
  ProductCollection(T,S) {is a base for} (T×tS)
  ⋃(T×tS) = ⋃T × ⋃S
proof -
  from A1 show
    (T×tS) {is a topology}
    ProductCollection(T,S) {is a base for} (T×tS)
    using Top_1_4_L1 ProductCollection_def
      SatisfiesBaseCondition_def ProductTopology_def Top_1_2_T1
    by auto
  then show ⋃(T×tS) = ⋃T × ⋃S
    using Top_1_2_L5 ZF1_1_L6 by simp
qed
```

Each point of a set open in the product topology has a neighborhood which is a cartesian product of open sets.

```
lemma prod_top_point_neighb:
  assumes A1: T {is a topology}  S {is a topology} and
  A2: U ∈ ProductTopology(T,S) and A3: x ∈ U
  shows ∃V W. V∈T ∧ W∈S ∧ V×W ⊆ U ∧ x ∈ V×W
proof -
  from A1 have
    ProductCollection(T,S) {is a base for} ProductTopology(T,S)
    using Top_1_4_T1 by simp
  with A2 A3 obtain Z where
    Z ∈ ProductCollection(T,S) and Z ⊆ U ∧ x∈Z
    using point_open_base_neigh by blast
  then obtain V W where V ∈ T and W∈S and V×W ⊆ U ∧ x ∈ V×W
    using ProductCollection_def by auto
  thus thesis by auto
qed
```

Products of open sets are open in the product topology.

```
lemma prod_open_open_prod:
  assumes A1: T {is a topology}  S {is a topology} and
  A2: U∈T V∈S
  shows U×V ∈ ProductTopology(T,S)
proof -
  from A1 have
    ProductCollection(T,S) {is a base for} ProductTopology(T,S)
    using Top_1_4_T1 by simp
  moreover from A2 have U×V ∈ ProductCollection(T,S)
    unfolding ProductCollection_def by auto
  ultimately show U×V ∈ ProductTopology(T,S)
    using base_sets_open by simp
qed
```

Sets that are open in the product topology are contained in the product of the carrier.

```
lemma prod_open_type: assumes A1: T {is a topology}  S {is a topology}
and
  A2: V ∈ ProductTopology(T,S)
  shows V ⊆ ⋃T × ⋃S
proof -
  from A2 have V ⊆ ⋃ ProductTopology(T,S) by auto
  with A1 show thesis using Top_1_4_T1 by simp
qed
```

A reverse of `prod_top_point_neighb`: if each point of set has an neighborhood in the set that is a cartesian product of open sets, then the set is open.

```
lemma point_neighb_prod_top:
  assumes T {is a topology}  S {is a topology}
  and ∀p∈V. ∃U∈T.∃W∈S. p∈U×W ∧ U×W ⊆ V
shows V ∈ ProductTopology(T,S)
proof -
  from assms(1,2) have I: topology0(ProductTopology(T,S))
    using Top_1_4_T1(1) topology0_def by simp
  moreover
  { fix p assume p∈V
    with assms(3) obtain U W where U∈T W∈S p∈U×W U×W ⊆ V
      by auto
    with assms(1,2) have ∃N∈ProductTopology(T,S). p∈N ∧ N⊆V
      using prod_open_open_prod by auto
  } hence ∀p∈V. ∃N∈ProductTopology(T,S). p∈N ∧ N⊆V by blast
  ultimately show thesis using topology0.open_neigh_open by simp
qed
```

Suppose we have subsets $A \subseteq X, B \subseteq Y$, where X, Y are topological spaces with topologies T, S. We can the consider relative topologies on T_A, S_B on sets A, B and the collection of cartesian products of sets open in T_A, S_B, (namely $\{U \times V : U \in T_A, V \in S_B\}$. The next lemma states that this collection is a base of the product topology on $X \times Y$ restricted to the product $A \times B$.

```
lemma prod_restr_base_restr:
  assumes A1: T {is a topology}  S {is a topology}
  shows
  ProductCollection(T {restricted to} A, S {restricted to} B)
  {is a base for} (ProductTopology(T,S) {restricted to} A×B)
proof -
  let ℬ = ProductCollection(T {restricted to} A, S {restricted to} B)
  let τ = ProductTopology(T,S)
  from A1 have (τ {restricted to} A×B) {is a topology}
    using Top_1_4_T1 topology0_def topology0.Top_1_L4
    by simp
  moreover have ℬ ⊆ (τ {restricted to} A×B)
```

```
  proof
    fix U assume U ∈ ℬ
    then obtain U_A U_B where U = U_A × U_B and
      U_A ∈ (T {restricted to} A) and U_B ∈ (S {restricted to} B)
      using ProductCollection_def by auto
    then obtain W_A W_B where
      W_A ∈ T  U_A = W_A ∩ A and W_B ∈ S  U_B = W_B ∩ B
      using RestrictedTo_def by auto
    with ‹U = U_A × U_B› have U = W_A×W_B ∩ (A×B) by auto
    moreover from A1 ‹W_A ∈ T› and ‹W_B ∈ S› have W_A×W_B ∈ τ
      using prod_open_open_prod by simp
    ultimately show U ∈ τ {restricted to} A×B
      using RestrictedTo_def by auto
  qed
  moreover have ∀U ∈ τ {restricted to} A×B.
    ∃C ∈ Pow(ℬ). U = ⋃C
  proof
    fix U assume U ∈ τ {restricted to} A×B
    then obtain W where W ∈ τ and U = W ∩ (A×B)
      using RestrictedTo_def by auto
    from A1 ‹W ∈ τ› obtain A_W  where
      A_W ∈ Pow(ProductCollection(T,S)) and W = ⋃A_W
       using Top_1_4_T1 IsAbaseFor_def by auto
    let C = {V ∩ A×B. V ∈ A_W}
    have C ∈ Pow(ℬ) and U = ⋃C
    proof -
      { fix R assume R ∈ C
 then obtain V where V ∈ A_W and R = V ∩ A×B
   by auto
 with ‹A_W ∈ Pow(ProductCollection(T,S))› obtain V_T V_S where
   V_T ∈ T and V_S ∈ S and V = V_T × V_S
   using ProductCollection_def by auto
 with ‹R = V ∩ A×B› have R ∈ ℬ
   using ProductCollection_def RestrictedTo_def
   by auto
      } then show C ∈ Pow(ℬ) by auto
      from ‹U = W ∩ (A×B)› and ‹W = ⋃A_W›
      show U = ⋃C by auto
    qed
    thus ∃C ∈ Pow(ℬ). U = ⋃C by blast
  qed
  ultimately show thesis by (rule is_a_base_criterion)
qed
```

We can commute taking restriction (relative topology) and product topology. The reason the two topologies are the same is that they have the same base.

```
lemma prod_top_restr_comm:
  assumes A1: T {is a topology}  S {is a topology}
  shows
```

```
  ProductTopology(T {restricted to} A,S {restricted to} B) =
  ProductTopology(T,S) {restricted to} (A×B)
proof -
  let ℬ = ProductCollection(T {restricted to} A, S {restricted to} B)
  from A1 have
    ℬ {is a base for} ProductTopology(T {restricted to} A,S {restricted
to} B)
    using topology0_def topology0.Top_1_L4 Top_1_4_T1 by simp
  moreover from A1 have
    ℬ {is a base for} ProductTopology(T,S) {restricted to} (A×B)
    using prod_restr_base_restr by simp
  ultimately show thesis by (rule same_base_same_top)
qed
```

Projection of a section of an open set is open.

```
lemma prod_sec_open1: assumes A1: T {is a topology}  S {is a topology}
and
  A2: V ∈ ProductTopology(T,S) and A3: x ∈ ⋃T
  shows {y ∈ ⋃S. ⟨x,y⟩ ∈ V} ∈ S
proof -
  let A = {y ∈ ⋃S. ⟨x,y⟩ ∈ V}
  from A1 have topology0(S) using topology0_def by simp
  moreover have ∀y∈A.∃W∈S. (y∈W ∧ W⊆A)
    proof
      fix y assume y ∈ A
      then have ⟨x,y⟩ ∈ V by simp
      with A1 A2 have ⟨x,y⟩ ∈ ⋃T × ⋃S using prod_open_type by blast
      hence x ∈ ⋃T and y ∈ ⋃S by auto
      from A1 A2 ‹⟨x,y⟩ ∈ V› have ∃U W. U∈T ∧ W∈S ∧ U×W ⊆ V ∧ ⟨x,y⟩
∈ U×W
        by (rule prod_top_point_neighb)
      then obtain U W where  U∈T W∈S U×W ⊆ V ⟨x,y⟩ ∈ U×W
        by auto
      with A1 A2 show ∃W∈S. (y∈W ∧ W⊆A) using prod_open_type section_proj
        by auto
    qed
  ultimately show thesis by (rule topology0.open_neigh_open)
qed
```

Projection of a section of an open set is open. This is dual of `prod_sec_open1` with a very similar proof.

```
lemma prod_sec_open2: assumes A1: T {is a topology}  S {is a topology}
and
  A2: V ∈ ProductTopology(T,S) and A3: y ∈ ⋃S
  shows {x ∈ ⋃T. ⟨x,y⟩ ∈ V} ∈ T
proof -
  let A = {x ∈ ⋃T. ⟨x,y⟩ ∈ V}
  from A1 have topology0(T) using topology0_def by simp
  moreover have ∀x∈A.∃W∈T. (x∈W ∧ W⊆A)
```

```
    proof
      fix x assume x ∈ A
      then have ⟨x,y⟩ ∈ V by simp
      with A1 A2 have ⟨x,y⟩ ∈ ⋃T × ⋃S using prod_open_type by blast
      hence x ∈ ⋃T and y ∈ ⋃S by auto
      from A1 A2 ‹⟨x,y⟩ ∈ V› have ∃U W. U∈T ∧ W∈S ∧ U×W ⊆ V ∧ ⟨x,y⟩
∈ U×W
        by (rule prod_top_point_neighb)
      then obtain U W where  U∈T W∈S U×W ⊆ V ⟨x,y⟩ ∈ U×W
        by auto
      with A1 A2 show ∃W∈T. (x∈W ∧ W⊆A) using prod_open_type section_proj
        by auto
    qed
  ultimately show thesis by (rule topology0.open_neigh_open)
qed
```

65.4 Hausdorff spaces

In this section we study properties of Hausdorff spaces (sometimes called separated spaces) These are topological spaces that are T_2 as defined above.

A space is Hausdorff if and only if the diagonal $\Delta = \{\langle x,x\rangle : x \in X\}$ is closed in the product topology on $X \times X$.

```
theorem t2_iff_diag_closed: assumes T {is a topology}
  shows T {is T₂} ⟷ {⟨x,x⟩. x∈⋃T} {is closed in} ProductTopology(T,T)
proof
  let X = ⋃T
  from assms(1) have I: topology0(ProductTopology(T,T))
    using Top_1_4_T1(1) topology0_def by simp
  assume T {is T₂} show {⟨x,x⟩. x∈X} {is closed in} ProductTopology(T,T)
  proof -
    let D_c = X×X - {⟨x,x⟩. x∈X}
    have ∀p∈D_c.∃U∈T.∃V∈T. p∈U×V ∧ U×V ⊆ D_c
    proof -
      { fix p assume p∈D_c
        then obtain x y where p=⟨x,y⟩ x∈X y∈X x≠y by auto
        with ‹T {is T₂}› obtain U V where U∈T V∈T x∈U y∈V U∩V = 0
          unfolding isT2_def by blast
        with assms ‹p=⟨x,y⟩› have ∃U∈T.∃V∈T. p∈U×V ∧ U×V ⊆ D_c by auto
      } hence ∀p. p∈D_c ⟶ (∃U∈T.∃V∈T. p∈U×V ∧ U×V ⊆ D_c) by simp
      then show thesis by (rule exists_in_set)
    qed
    with assms show thesis using Top_1_4_T1(3) point_neighb_prod_top

      unfolding IsClosed_def by auto
  qed
next
  let X = ⋃T
```

```
  assume A: {⟨x,x⟩. x∈X} {is closed in} ProductTopology(T,T) show T {is T₂}
  proof -
    { let D_c = X×X - {⟨x,x⟩. x∈X}
      fix x y assume x∈X y∈X x≠y
      with assms A have D_c ∈ ProductTopology(T,T) and ⟨x,y⟩ ∈ D_c
        using Top_1_4_T1(3) unfolding IsClosed_def by auto
      with assms obtain U V where U∈T V∈T U×V ⊆ D_c ⟨x,y⟩ ∈ U×V
        using prod_top_point_neighb by blast
      moreover from ‹U×V ⊆ D_c› have U∩V = 0 by auto
      ultimately have ∃U∈T.∃V∈T. x∈U ∧ y∈V ∧ U∩V=0 by auto
    } then show T {is T₂} unfolding isT2_def by simp
  qed
qed

end
```

66 Topology 2

```
theory Topology_ZF_2 imports Topology_ZF_1 func1 Fol1

begin
```

This theory continues the series on general topology and covers the definition and basic properties of continuous functions. We also introduce the notion of homeomorphism an prove the pasting lemma.

66.1 Continuous functions.

In this section we define continuous functions and prove that certain conditions are equivalent to a function being continuous.

In standard math we say that a function is continuous with respect to two topologies τ_1, τ_2 if the inverse image of sets from topology τ_2 are in τ_1. Here we define a predicate that is supposed to reflect that definition, with a difference that we don't require in the definition that τ_1, τ_2 are topologies. This means for example that when we define measurable functions, the definition will be the same.
The notation f-(A) means the inverse image of (a set) A with respect to (a function) f.

```
definition
  IsContinuous(τ₁,τ₂,f) ≡ (∀U∈τ₂. f-(U) ∈ τ₁)
```

The space of continuous functions mapping $X = \bigcup \tau_1$ to $Y = \bigcup \tau_2$ will be denoted Cont(τ_1,τ_2).

definition

```
Cont(τ1,τ2) ≡ {f∈(⋃τ1)→(⋃τ2). IsContinuous(τ1,τ2,f)}
```

A trivial example of a continuous function - identity is continuous.

```
lemma id_cont: shows IsContinuous(τ,τ,id(⋃τ))
proof -
  { fix U assume U∈τ
    then have id(⋃τ)-(U) = U using vimage_id_same by auto
    with ‹U∈τ› have id(⋃τ)-(U) ∈ τ by simp
  } then show IsContinuous(τ,τ,id(⋃τ)) unfolding IsContinuous_def
    by simp
qed
```

Identity is in the space of continuous functions from $\bigcup \tau$ to itself.

```
lemma id_cont_sp: shows {⟨x,x⟩. x∈⋃τ} ∈ Cont(τ,τ)
proof -
  have id(⋃τ) : ⋃τ → ⋃τ and IsContinuous(τ,τ,id(⋃τ))
    using id_type id_cont by auto
  moreover have id(⋃τ) = {⟨x,x⟩. x∈⋃τ} by blast
  ultimately show thesis unfolding Cont_def by simp
qed
```

A constant function is continuous.

```
lemma const_cont: assumes T {is a topology}
  shows IsContinuous(T,τ,ConstantFunction(⋃T,c))
proof -
   let C = ConstantFunction(⋃T,c)
   { fix U assume U∈τ
    have C-(U) ∈ T
    proof -
      { assume c∈U
        with assms have C-(U) ∈ T  using carr_open const_vimage_domain

          by simp
      }
      moreover
      { assume c∉U
        with assms have C-(U) ∈ T using empty_open const_vimage_empty

          by simp
      }
      ultimately show C-(U) ∈ T by auto
    qed
  } then show thesis unfolding IsContinuous_def
    by simp
qed
```

If $c \in Y = \bigcup S$, then the constant function defined on $X = \bigcup T$ that is equal to c is in the the space of continuous functions from X to Y.

```
lemma const_cont_sp: assumes T {is a topology} c∈⋃S
```

```
  shows {⟨x,c⟩. x∈⋃T} ∈ Cont(T,S)
  using assms ZF_fun_from_total const_fun_def_alt const_cont
  unfolding Cont_def by simp
```

We will work with a pair of topological spaces. The following locale sets up our context that consists of two topologies τ_1, τ_2 and a continuous function $f : X_1 \to X_2$, where X_i is defined as $\bigcup \tau_i$ for $i = 1, 2$. We also define notation `cl₁(A)` and `cl₂(A)` for closure of a set A in topologies τ_1 and τ_2, respectively.

```
locale two_top_spaces0 =

  fixes τ₁
  assumes tau1_is_top: τ₁ {is a topology}

  fixes τ₂
  assumes tau2_is_top: τ₂ {is a topology}

  fixes X₁
  defines X1_def [simp]: X₁ ≡ ⋃τ₁

  fixes X₂
  defines X2_def [simp]: X₂ ≡ ⋃τ₂

  fixes f
  assumes fmapAssum: f: X₁ → X₂

  fixes isContinuous (_ {is continuous} [50] 50)
  defines isContinuous_def [simp]: g {is continuous} ≡ IsContinuous(τ₁,τ₂,g)

  fixes cl₁
  defines cl1_def [simp]: cl₁(A) ≡ Closure(A,τ₁)

  fixes cl₂
  defines cl2_def [simp]: cl₂(A) ≡ Closure(A,τ₂)
```

First we show that theorems proven in locale `topology0` are valid when applied to topologies τ_1 and τ_2.

```
lemma (in two_top_spaces0) topol_cntxs_valid:
  shows topology0(τ₁) and topology0(τ₂)
  using tau1_is_top tau2_is_top topology0_def by auto
```

For continuous functions the inverse image of a closed set is closed.

```
lemma (in two_top_spaces0) TopZF_2_1_L1:
  assumes A1: f {is continuous} and A2: D {is closed in} τ₂
  shows f-(D) {is closed in} τ₁
proof -
  from fmapAssum have  f-(D) ⊆ X₁ using func1_1_L3 by simp
  moreover from fmapAssum have f-(X₂ - D) =  X₁ - f-(D)
```

```
    using Pi_iff function_vimage_Diff func1_1_L4 by auto
  ultimately have X1 - f-(X2 - D) = f-(D) by auto
  moreover from A1 A2 have (X1 - f-(X2 - D)) {is closed in} τ1
    using IsClosed_def IsContinuous_def topol_cntxs_valid topology0.Top_3_L9
    by simp
  ultimately show f-(D) {is closed in} τ1 by simp
qed
```

If the inverse image of every closed set is closed, then the image of a closure is contained in the closure of the image.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L2:
  assumes A1: ∀D. ((D {is closed in} τ2) ⟶ f-(D) {is closed in} τ1)
  and A2: A ⊆ X1
  shows f(cl1(A)) ⊆ cl2(f(A))
proof -
  from fmapAssum have f(A) ⊆ cl2(f(A))
    using func1_1_L6 topol_cntxs_valid topology0.cl_contains_set
    by simp
  with fmapAssum have f-(f(A)) ⊆ f-(cl2(f(A)))
    by auto
  moreover from fmapAssum A2 have A ⊆ f-(f(A))
    using func1_1_L9 by simp
  ultimately have A ⊆ f-(cl2(f(A))) by auto
  with fmapAssum A1 have f(cl1(A)) ⊆ f(f-(cl2(f(A))))
    using func1_1_L6 func1_1_L8 IsClosed_def
      topol_cntxs_valid topology0.cl_is_closed topology0.Top_3_L13
    by simp
  moreover from fmapAssum have f(f-(cl2(f(A)))) ⊆ cl2(f(A))
    using fun_is_function function_image_vimage by simp
  ultimately show f(cl1(A)) ⊆ cl2(f(A))
    by auto
qed
```

If $f\left(\overline{A}\right) \subseteq \overline{f(A)}$ (the image of the closure is contained in the closure of the image), then $\overline{f^{-1}(B)} \subseteq f^{-1}\left(\overline{B}\right)$ (the inverse image of the closure contains the closure of the inverse image).

```
lemma (in two_top_spaces0) Top_ZF_2_1_L3:
  assumes A1: ∀ A. ( A ⊆ X1 ⟶ f(cl1(A)) ⊆ cl2(f(A)))
  shows ∀B. ( B ⊆ X2 ⟶ cl1(f-(B)) ⊆ f-(cl2(B)) )
proof -
  { fix B assume B ⊆ X2
    from fmapAssum A1 have f(cl1(f-(B))) ⊆ cl2(f(f-(B)))
      using func1_1_L3 by simp
    moreover from fmapAssum ‹B ⊆ X2› have cl2(f(f-(B))) ⊆ cl2(B)
      using fun_is_function function_image_vimage func1_1_L6
 topol_cntxs_valid topology0.top_closure_mono
      by simp
    ultimately have f-(f(cl1(f-(B)))) ⊆ f-(cl2(B))
      using fmapAssum fun_is_function by auto
```

```
    moreover from fmapAssum ‹B ⊆ X₂› have
      cl₁(f-(B)) ⊆ f-(f(cl₁(f-(B))))
      using func1_1_L3 func1_1_L9 IsClosed_def
 topol_cntxs_valid topology0.cl_is_closed by simp
    ultimately have cl₁(f-(B)) ⊆ f-(cl₂(B)) by auto
  } then show thesis by simp
qed
```

If $\overline{f^{-1}(B)} \subseteq f^{-1}\left(\overline{B}\right)$ (the inverse image of a closure contains the closure of the inverse image), then the function is continuous. This lemma closes a series of implications in lemmas `Top_ZF_2_1_L1`, `Top_ZF_2_1_L2` and `Top_ZF_2_1_L3` showing equivalence of four definitions of continuity.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L4:
  assumes A1: ∀B. ( B ⊆ X₂ ⟶ cl₁(f-(B)) ⊆ f-(cl₂(B)) )
  shows f {is continuous}
proof -
  { fix U assume U ∈ τ₂
    then have (X₂ - U) {is closed in} τ₂
      using topol_cntxs_valid topology0.Top_3_L9 by simp
    moreover have X₂ - U ⊆ ⋃τ₂ by auto
    ultimately have cl₂(X₂ - U) = X₂ - U
      using topol_cntxs_valid topology0.Top_3_L8 by simp
    moreover from A1 have cl₁(f-(X₂ - U)) ⊆ f-(cl₂(X₂ - U))
      by auto
    ultimately have cl₁(f-(X₂ - U)) ⊆ f-(X₂ - U) by simp
    moreover from fmapAssum have f-(X₂ - U) ⊆ cl₁(f-(X₂ - U))
      using func1_1_L3 topol_cntxs_valid topology0.cl_contains_set
      by simp
    ultimately have f-(X₂ - U) {is closed in} τ₁
      using fmapAssum func1_1_L3 topol_cntxs_valid topology0.Top_3_L8
      by auto
    with fmapAssum have f-(U) ∈ τ₁
      using fun_is_function function_vimage_Diff func1_1_L4
 func1_1_L3 IsClosed_def double_complement by simp
  } then have ∀U∈τ₂. f-(U) ∈ τ₁ by simp
  then show thesis using IsContinuous_def by simp
qed
```

For continuous functions the closure of the inverse image is contained in the inverse image of the closure. This is a shortcut through a series of implications provided by `TopZF_2_1_L1`, `Top_ZF_2_1_L2` and `Top_ZF_2_1_L3`.

```
corollary (in two_top_spaces0) im_cl_in_cl_im:
  assumes f {is continuous} and B ⊆ X₂
  shows cl₁(f-(B)) ⊆ f-(cl₂(B))
  using assms TopZF_2_1_L1 Top_ZF_2_1_L2 Top_ZF_2_1_L3 by simp
```

Another condition for continuity: it is sufficient to check if the inverse image of every set in a base is open.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L5:
  assumes A1: B {is a base for} τ₂ and A2: ∀U∈B. f-(U) ∈ τ₁
  shows f {is continuous}
proof -
  { fix V assume A3: V ∈ τ₂
    with A1 obtain A where A ⊆ B  V = ⋃A
      using IsAbaseFor_def by auto
    with A2 have {f-(U). U∈A} ⊆ τ₁ by auto
    with tau1_is_top have ⋃ {f-(U). U∈A} ∈ τ₁
      using IsATopology_def by simp
    moreover from ‹A ⊆ B› ‹V = ⋃A› have f-(V) = ⋃{f-(U). U∈A}
      by auto
    ultimately have f-(V) ∈  τ₁ by simp
  } then show f {is continuous} using IsContinuous_def
    by simp
qed
```

We can strenghten the previous lemma: it is sufficient to check if the inverse image of every set in a subbase is open. The proof is rather awkward, as usual when we deal with general intersections. We have to keep track of the case when the collection is empty.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L6:
  assumes A1: B {is a subbase for} τ₂ and A2: ∀U∈B. f-(U) ∈ τ₁
  shows f {is continuous}
proof -
  let C = {⋂A. A ∈ FinPow(B)}
  from A1 have C {is a base for} τ₂
    using IsAsubBaseFor_def by simp
  moreover have ∀U∈C. f-(U) ∈ τ₁
  proof
    fix U assume U∈C
    { assume f-(U) = 0
      with tau1_is_top have f-(U) ∈ τ₁
 using empty_open by simp }
    moreover
    { assume f-(U) ≠ 0
      then have U≠0 by (rule func1_1_L13)
      moreover from ‹U∈C› obtain A where
 A ∈ FinPow(B) and U = ⋂A
 by auto
      ultimately have ⋂A≠0 by simp
      then have A≠0 by (rule inter_nempty_nempty)
      then have {f-(W). W∈A} ≠ 0 by simp
      moreover from A2 ‹A ∈ FinPow(B)› have {f-(W). W∈A} ∈ FinPow(τ₁)
 by (rule fin_image_fin)
      ultimately have ⋂{f-(W). W∈A} ∈ τ₁
 using topol_cntxs_valid topology0.fin_inter_open_open by simp
      moreover
      from ‹A ∈ FinPow(B)› have A ⊆ B using FinPow_def by simp
```

```
      with tau2_is_top A1 have A ⊆ Pow(X₂)
 using IsAsubBaseFor_def IsATopology_def by auto
      with fmapAssum ‹A≠0› ‹U = ⋂A› have f-(U) = ⋂{f-(W). W∈A}
 using func1_1_L12 by simp
      ultimately have f-(U) ∈ τ₁ by simp }
    ultimately show f-(U) ∈ τ₁ by blast
  qed
  ultimately show f {is continuous}
    using Top_ZF_2_1_L5 by simp
qed
```

A dual of `Top_ZF_2_1_L5`: a function that maps base sets to open sets is open.

```
lemma (in two_top_spaces0) base_image_open:
  assumes A1: ℬ {is a base for} τ₁ and A2: ∀B∈ℬ. f(B) ∈ τ₂ and A3:
U∈τ₁
  shows f(U) ∈ τ₂
proof -
  from A1 A3 obtain ℰ where ℰ ∈ Pow(ℬ) and U = ⋃ℰ using Top_1_2_L1
by blast
  with A1 have f(U) = ⋃{f(E). E ∈ ℰ} using Top_1_2_L5  fmapAssum image_of_Union
    by auto
  moreover
  from A2 ‹ℰ ∈ Pow(ℬ)› have {f(E). E ∈ ℰ} ∈ Pow(τ₂) by auto
  then have ⋃{f(E). E ∈ ℰ} ∈ τ₂ using tau2_is_top IsATopology_def by
simp
  ultimately show thesis using tau2_is_top IsATopology_def by auto
qed
```

A composition of two continuous functions is continuous.

```
lemma comp_cont: assumes IsContinuous(T,S,f) and IsContinuous(S,R,g)
  shows IsContinuous(T,R,g O f)
  using assms IsContinuous_def vimage_comp by simp
```

A composition of three continuous functions is continuous.

```
lemma comp_cont3:
  assumes IsContinuous(T,S,f) and IsContinuous(S,R,g) and IsContinuous(R,P,h)
  shows IsContinuous(T,P,h O g O f)
  using assms IsContinuous_def vimage_comp by simp
```

The graph of a continuous function into a Hausdorff topological space is closed in the product topology. Recall that in ZF a function is the same as its graph.

```
lemma (in two_top_spaces0) into_T2_graph_closed:
  assumes f {is continuous} τ₂ {is T₂}
  shows f {is closed in} ProductTopology(τ₁,τ₂)
proof -
  from fmapAssum have f = {⟨x,f(x)⟩. x∈X₁} using fun_is_set_of_pairs
    by simp
```

```
  let fc = X1×X2 - f
  have fc ∈ ProductTopology(τ1,τ2)
  proof -
    { fix p assume p∈fc
      then have p ∈ X1×X2 and p ∉ f by auto
      from ‹p∈X1×X2› obtain x y where x∈X1 y∈X2 p = ⟨x,y⟩
        by auto
      have y≠f(x)
      proof -
        { assume y=f(x)
          with ‹x∈X1› ‹p = ⟨x,y⟩› have p ∈ {⟨x,f(x)⟩. x∈X1} by auto
          with ‹f = {⟨x,f(x)⟩. x∈X1}› ‹p ∉ f› have False by auto
        } thus y≠f(x) by auto
      qed
      from fmapAssum ‹x∈X1› have f(x) ∈ X2 by (rule apply_funtype)
      with ‹y∈X2›  have y∈⋃τ2 f(x) ∈ ⋃τ2 by auto
      with assms(2) ‹y≠f(x)› obtain U V
        where U∈τ2 V∈τ2 y∈U f(x)∈V U∩V = 0
        unfolding isT2_def by blast
      let W = f-(V)
      have W∈τ1 W⊆X1 U⊆X2 x∈W p∈W×U f(W) ⊆ V
      proof -
        from assms(1) have IsContinuous(τ1,τ2,f) by simp
        with ‹V∈τ2› ‹U∈τ2› show W∈τ1 W⊆X1 U⊆X2
          unfolding IsContinuous_def by auto
        from fmapAssum ‹x∈X1› ‹f(x)∈V› show x∈W using func1_1_L15
          by auto
        with ‹y∈U› ‹y∈U› ‹p = ⟨x,y⟩› show  p∈W×U by simp
        from fmapAssum show f(W) ⊆ V
          using fun_is_fun function_image_vimage by simp
      qed
      from fmapAssum ‹U∩V = 0› ‹W⊆X1› ‹U⊆X2› have W×U ⊆ fc
        using vimage_prod_dis_graph by blast
      with ‹W∈τ1› ‹U∈τ2› ‹p∈W×U› have ∃W∈τ1.∃U∈τ2. p∈W×U ∧ W×U ⊆
fc
        by blast
    }
    with tau1_is_top tau2_is_top show fc ∈ ProductTopology(τ1,τ2)
      using point_neighb_prod_top by simp
  qed
  with fmapAssum tau1_is_top tau2_is_top show thesis
    using fun_subset_prod Top_1_4_T1(3) unfolding IsClosed_def
    by auto
qed
```

66.2 Homeomorphisms

This section studies "homeomorphisms" - continous bijections whose inverses are also continuous. Notions that are preserved by (commute with) homeomorphisms are called ”topological invariants”.

Homeomorphism is a bijection that preserves open sets.

```
definition IsAhomeomorphism(T,S,f) ≡
        f ∈ bij(⋃T,⋃S) ∧ IsContinuous(T,S,f) ∧ IsContinuous(S,T,converse(f))
```

Inverse (converse) of a homeomorphism is a homeomorphism.

```
lemma homeo_inv: assumes IsAhomeomorphism(T,S,f)
  shows IsAhomeomorphism(S,T,converse(f))
  using assms IsAhomeomorphism_def bij_converse_bij bij_converse_converse
    by auto
```

Homeomorphisms are open maps.

```
lemma homeo_open: assumes IsAhomeomorphism(T,S,f) and U∈T
  shows f(U) ∈ S
  using assms image_converse IsAhomeomorphism_def IsContinuous_def by
simp
```

A continuous bijection that is an open map is a homeomorphism.

```
lemma bij_cont_open_homeo:
  assumes f ∈ bij(⋃T,⋃S) and IsContinuous(T,S,f) and ∀U∈T. f(U) ∈
S
  shows IsAhomeomorphism(T,S,f)
  using assms image_converse IsAhomeomorphism_def IsContinuous_def by
auto
```

A continuous bijection that maps base to open sets is a homeomorphism.

```
lemma (in two_top_spaces0) bij_base_open_homeo:
  assumes A1: f ∈ bij(X₁,X₂) and A2: ℬ {is a base for} τ₁  and A3: 𝒞
{is a base for} τ₂ and
  A4: ∀U∈𝒞. f-(U) ∈ τ₁ and A5: ∀V∈ℬ. f(V) ∈ τ₂
  shows IsAhomeomorphism(τ₁,τ₂,f)
  using assms tau2_is_top tau1_is_top bij_converse_bij bij_is_fun two_top_spaces0_def

  image_converse two_top_spaces0.Top_ZF_2_1_L5 IsAhomeomorphism_def by
simp
```

A bijection that maps base to base is a homeomorphism.

```
lemma (in two_top_spaces0) bij_base_homeo:
  assumes A1: f ∈ bij(X₁,X₂) and A2: ℬ {is a base for} τ₁ and
  A3: {f(B). B∈ℬ} {is a base for} τ₂
  shows IsAhomeomorphism(τ₁,τ₂,f)
proof -
  note A1
  moreover have f {is continuous}
  proof -
    { fix C assume C ∈ {f(B). B∈ℬ}
      then obtain B where B∈ℬ and I: C = f(B) by auto
```

```
      with A2 have B ⊆ X_1 using Top_1_2_L5 by auto
      with A1 A2 ‹B∈ℬ› I have f-(C) ∈ τ_1
        using bij_def inj_vimage_image base_sets_open by auto
    } hence ∀C ∈ {f(B). B∈ℬ}. f-(C) ∈ τ_1 by auto
    with A3 show thesis by (rule Top_ZF_2_1_L5)
  qed
  moreover
  from A3 have ∀B∈ℬ. f(B) ∈ τ_2 using base_sets_open by auto
  with A2 have ∀U∈τ_1. f(U) ∈ τ_2 using base_image_open by simp
  ultimately show thesis using bij_cont_open_homeo by simp
qed
```

Interior is a topological invariant.

```
theorem int_top_invariant: assumes A1: A⊆⋃T and A2: IsAhomeomorphism(T,S,f)
  shows f(Interior(A,T)) = Interior(f(A),S)
proof -
  let 𝒜 = {U∈T. U⊆A}
  have I: {f(U). U∈𝒜} = {V∈S. V ⊆ f(A)}
  proof
    from A2 show {f(U). U∈𝒜} ⊆ {V∈S. V ⊆ f(A)}
      using homeo_open by auto
    { fix V assume V ∈ {V∈S. V ⊆ f(A)}
      hence V∈S and II: V ⊆ f(A) by auto
      let U = f-(V)
      from II have U ⊆ f-(f(A)) by auto
      moreover from assms have f-(f(A)) = A
        using IsAhomeomorphism_def bij_def inj_vimage_image by auto
      moreover from A2 ‹V∈S› have U∈T
        using IsAhomeomorphism_def IsContinuous_def by simp
      moreover
      from ‹V∈S› have V ⊆ ⋃S by auto
      with A2 have V = f(U)
        using IsAhomeomorphism_def bij_def surj_image_vimage by auto
      ultimately have V ∈ {f(U). U∈𝒜} by auto
    } thus {V∈S. V ⊆ f(A)} ⊆ {f(U). U∈𝒜} by auto
  qed
  have f(Interior(A,T)) =  f(⋃𝒜) unfolding Interior_def by simp
  also from A2 have ... = ⋃{f(U). U∈𝒜}
    using IsAhomeomorphism_def bij_def inj_def image_of_Union by auto
  also from I have ... = Interior(f(A),S) unfolding Interior_def by simp
  finally show thesis by simp
qed
```

66.3 Topologies induced by mappings

In this section we consider various ways a topology may be defined on a set that is the range (or the domain) of a function whose domain (or range) is a topological space.

A bijection from a topological space induces a topology on the range.

```
theorem bij_induced_top: assumes A1: T {is a topology} and A2: f ∈ bij(⋃T,Y)
  shows
  {f(U). U∈T} {is a topology} and
  { {f(x).x∈U}. U∈T} {is a topology} and
  (⋃{f(U). U∈T}) = Y and
  IsAhomeomorphism(T, {f(U). U∈T},f)
proof -
  from A2 have f ∈ inj(⋃T,Y) using bij_def by simp
  then have f:⋃T→Y using inj_def by simp
  let S = {f(U). U∈T}
  { fix M assume M ∈ Pow(S)
    let M_T = {f-(V). V∈M}
    have M_T ⊆ T
    proof
      fix W assume W∈M_T
      then obtain V where V∈M and I: W = f-(V) by auto
      with ‹M ∈ Pow(S)› have V∈S by auto
      then obtain U where U∈T and  V = f(U) by auto
      with I have W = f-(f(U)) by simp
      with ‹f ∈ inj(⋃T,Y)›  ‹U∈T› have W = U using inj_vimage_image
by blast
      with ‹U∈T› show W∈T by simp
    qed
    with A1 have (⋃M_T) ∈ T using IsATopology_def by simp
    hence f(⋃M_T) ∈  S by auto
    moreover have f(⋃M_T) = ⋃M
    proof -
      from ‹f:⋃T→Y› ‹M_T ⊆ T›  have f(⋃M_T) = ⋃{f(U). U∈M_T}
         using image_of_Union by auto
      moreover have {f(U). U∈M_T} = M
      proof -
        from ‹f:⋃T→Y› have ∀U∈T. f(U) ⊆ Y using  func1_1_L6 by simp
        with ‹M ∈ Pow(S)› have M ⊆ Pow(Y) by auto
        with A2 show {f(U). U∈M_T} = M using bij_def surj_subsets by
auto
      qed
      ultimately show f(⋃M_T) = ⋃M by simp
    qed
    ultimately have ⋃M ∈ S by auto
  } then have ∀M∈Pow(S). ⋃M ∈ S by auto
  moreover
  { fix U V assume U∈S V∈S
    then obtain U_T V_T where U_T ∈ T   V_T ∈ T and
      I: U = f(U_T)  V = f(V_T)
      by auto
    with A1 have U_T∩V_T ∈ T using IsATopology_def by simp
    hence f(U_T∩V_T) ∈ S by auto
    moreover have f(U_T∩V_T) = U∩V
```

```
    proof -
      from ‹U_T ∈ T›  ‹V_T ∈ T› have U_T ⊆ ⋃T  V_T ⊆ ⋃T
        using bij_def by auto
      with ‹f ∈ inj(⋃T,Y)› I show f(U_T∩V_T) = U∩V using inj_image_inter

      by simp
    qed
    ultimately have U∩V ∈ S by simp
  } then have ∀U∈S. ∀V∈S. U∩V ∈ S by auto
  ultimately show S {is a topology} using IsATopology_def by simp
  moreover from ‹f:⋃T→Y› have ∀U∈T. f(U) = {f(x).x∈U}
    using func_imagedef by blast
  ultimately show { {f(x).x∈U}. U∈T} {is a topology} by simp
  show ⋃S =  Y
  proof
    from ‹f:⋃T→Y› have ∀U∈T. f(U) ⊆ Y using func1_1_L6 by simp
    thus ⋃S ⊆ Y by auto
    from A1 have f(⋃T) ⊆ ⋃S using IsATopology_def by auto
    with A2 show Y ⊆ ⋃S using bij_def surj_range_image_domain
      by auto
  qed
  show IsAhomeomorphism(T,S,f)
  proof -
    from A2  ‹⋃S =  Y› have f ∈ bij(⋃T,⋃S) by simp
    moreover have IsContinuous(T,S,f)
    proof -
      { fix V assume V∈S
        then obtain U where U∈T and V = f(U) by auto
        hence U ⊆ ⋃T and f-(V) = f-(f(U))  by auto
        with ‹f ∈ inj(⋃T,Y)›  ‹U∈T› have f-(V) ∈ T  using inj_vimage_image

          by simp
      } then show IsContinuous(T,S,f) unfolding IsContinuous_def by auto
    qed
    ultimately showIsAhomeomorphism(T,S,f) using bij_cont_open_homeo
      by auto
  qed
qed
```

66.4 Partial functions and continuity

Suppose we have two topologies τ_1, τ_2 on sets $X_i = \bigcup \tau_i, i = 1,2$. Consider some function $f : A \to X_2$, where $A \subseteq X_1$ (we will call such function "partial"). In such situation we have two natural possibilities for the pairs of topologies with respect to which this function may be continuous. One is obviously the original τ_1, τ_2 and in the second one the first element of the pair is the topology relative to the domain of the function: $\{A \cap U | U \in \tau_1\}$. These two possibilities are not exactly the same and the goal of this section is to explore the differences.

If a function is continuous, then its restriction is continous in relative topology.

```
lemma (in two_top_spaces0) restr_cont:
  assumes A1: A ⊆ X₁ and A2: f {is continuous}
  shows IsContinuous(τ₁ {restricted to} A, τ₂,restrict(f,A))
proof -
  let g = restrict(f,A)
  { fix U assume U ∈ τ₂
    with A2 have f-(U) ∈ τ₁ using IsContinuous_def by simp
    moreover from A1 have g-(U) = f-(U) ∩ A
      using fmapAssum func1_2_L1 by simp
    ultimately have g-(U) ∈ (τ₁ {restricted to} A)
      using RestrictedTo_def by auto
  } then show thesis using IsContinuous_def by simp
qed
```

If a function is continuous, then it is continuous when we restrict the topology on the range to the image of the domain.

```
lemma (in two_top_spaces0) restr_image_cont:
  assumes A1: f {is continuous}
  shows IsContinuous(τ₁, τ₂ {restricted to} f(X₁),f)
proof -
  have ∀U ∈ τ₂ {restricted to} f(X₁). f-(U) ∈ τ₁
  proof
    fix U assume U ∈ τ₂ {restricted to} f(X₁)
    then obtain V where V ∈ τ₂ and U = V ∩ f(X₁)
      using RestrictedTo_def by auto
    with A1 show  f-(U) ∈ τ₁
      using fmapAssum inv_im_inter_im IsContinuous_def
      by simp
  qed
  then show thesis using IsContinuous_def by simp
qed
```

A combination of `restr_cont` and `restr_image_cont`.

```
lemma (in two_top_spaces0) restr_restr_image_cont:
  assumes A1: A ⊆ X₁ and A2: f {is continuous} and
  A3: g = restrict(f,A) and
  A4: τ₃ = τ₁ {restricted to} A
  shows IsContinuous(τ₃, τ₂ {restricted to} g(A),g)
proof -
  from A1 A4 have ⋃τ₃ = A
    using union_restrict by auto
  have two_top_spaces0(τ₃, τ₂, g)
  proof -
    from A4 have
```

```
      τ₃ {is a topology} and τ₂ {is a topology}
      using tau1_is_top tau2_is_top
 topology0_def topology0.Top_1_L4 by auto
    moreover from A1 A3 ‹⋃τ₃ = A› have g: ⋃τ₃ → ⋃τ₂
      using fmapAssum restrict_type2 by simp
    ultimately show thesis using two_top_spaces0_def
      by simp
  qed
  moreover from assms have IsContinuous(τ₃, τ₂, g)
    using restr_cont by simp
  ultimately have IsContinuous(τ₃, τ₂ {restricted to} g(⋃τ₃),g)
    by (rule two_top_spaces0.restr_image_cont)
  moreover note ‹⋃τ₃ = A›
  ultimately show thesis by simp
qed
```

We need a context similar to two_top_spaces0 but without the global function $f : X_1 \to X_2$.

```
locale two_top_spaces1 =

  fixes τ₁
  assumes tau1_is_top: τ₁ {is a topology}

  fixes τ₂
  assumes tau2_is_top: τ₂ {is a topology}

  fixes X₁
  defines X1_def [simp]: X₁ ≡ ⋃τ₁

  fixes X₂
  defines X2_def [simp]: X₂ ≡ ⋃τ₂
```

If a partial function $g : X_1 \supseteq A \to X_2$ is continuous with respect to (τ_1, τ_2), then A is open (in τ_1) and the function is continuous in the relative topology.

```
lemma (in two_top_spaces1) partial_fun_cont:
  assumes A1: g:A→X₂ and A2: IsContinuous(τ₁,τ₂,g)
  shows A ∈ τ₁ and IsContinuous(τ₁ {restricted to} A, τ₂, g)
proof -
  from A2 have g-(X₂) ∈ τ₁
    using tau2_is_top IsATopology_def IsContinuous_def by simp
  with A1 show A ∈ τ₁ using func1_1_L4 by simp
  { fix V assume V ∈ τ₂
    with A2 have g-(V) ∈ τ₁ using IsContinuous_def by simp
    moreover
    from A1 have g-(V) ⊆ A using func1_1_L3 by simp
    hence g-(V) = A ∩ g-(V) by auto
    ultimately have g-(V) ∈ (τ₁ {restricted to} A)
      using RestrictedTo_def by auto
  } then show IsContinuous(τ₁ {restricted to} A, τ₂, g)
```

```
    using IsContinuous_def by simp
qed
```

For partial function defined on open sets continuity in the whole and relative topologies are the same.

```
lemma (in two_top_spaces1) part_fun_on_open_cont:
  assumes A1: g:A→X₂ and A2: A ∈ τ₁
  shows IsContinuous(τ₁,τ₂,g) ⟷
        IsContinuous(τ₁ {restricted to} A, τ₂, g)
proof
  assume IsContinuous(τ₁,τ₂,g)
  with A1 show IsContinuous(τ₁ {restricted to} A, τ₂, g)
    using partial_fun_cont by simp
  next
    assume I: IsContinuous(τ₁ {restricted to} A, τ₂, g)
    { fix V assume V ∈ τ₂
      with I have g-(V) ∈ (τ₁ {restricted to} A)
        using IsContinuous_def by simp
      then obtain W where W ∈ τ₁ and g-(V) = A∩W
        using RestrictedTo_def by auto
      with A2 have g-(V) ∈ τ₁ using tau1_is_top IsATopology_def
        by simp
    } then show IsContinuous(τ₁,τ₂,g) using IsContinuous_def
      by simp
qed
```

66.5 Product topology and continuity

We start with three topological spaces $(\tau_1, X_1), (\tau_2, X_2)$ and (τ_3, X_3) and a function $f : X_1 \times X_2 \to X_3$. We will study the properties of f with respect to the product topology $\tau_1 \times \tau_2$ and τ_3. This situation is similar as in locale `two_top_spaces0` but the first topological space is assumed to be a product of two topological spaces.

First we define a locale with three topological spaces.

```
locale prod_top_spaces0 =

  fixes τ₁
  assumes tau1_is_top: τ₁ {is a topology}

  fixes τ₂
  assumes tau2_is_top: τ₂ {is a topology}

  fixes τ₃
  assumes tau3_is_top: τ₃ {is a topology}

  fixes X₁
  defines X1_def [simp]: X₁ ≡ ⋃τ₁
```

```
  fixes X₂
  defines X2_def [simp]: X₂ ≡ ⋃τ₂

  fixes X₃
  defines X3_def [simp]: X₃ ≡ ⋃τ₃

  fixes η
  defines eta_def [simp]: η ≡ ProductTopology(τ₁,τ₂)
```

Fixing the first variable in a two-variable continuous function results in a continuous function.

```
lemma (in prod_top_spaces0) fix_1st_var_cont:
  assumes f: X₁×X₂→X₃ and IsContinuous(η,τ₃,f)
  and x∈X₁
  shows IsContinuous(τ₂,τ₃,Fix1stVar(f,x))
  using assms fix_1st_var_vimage IsContinuous_def tau1_is_top tau2_is_top
    prod_sec_open1 by simp
```

Fixing the second variable in a two-variable continuous function results in a continuous function.

```
lemma (in prod_top_spaces0) fix_2nd_var_cont:
  assumes f: X₁×X₂→X₃ and IsContinuous(η,τ₃,f)
  and y∈X₂
  shows IsContinuous(τ₁,τ₃,Fix2ndVar(f,y))
  using assms fix_2nd_var_vimage IsContinuous_def tau1_is_top tau2_is_top
    prod_sec_open2 by simp
```

Having two constinuous mappings we can construct a third one on the cartesian product of the domains.

```
lemma cart_prod_cont:
  assumes A1: τ₁ {is a topology} τ₂ {is a topology} and
  A2: η₁ {is a topology} η₂ {is a topology} and
  A3a: f₁:⋃τ₁→⋃η₁  and A3b: f₂:⋃τ₂→⋃η₂ and
  A4: IsContinuous(τ₁,η₁,f₁) IsContinuous(τ₂,η₂,f₂) and
  A5: g = {⟨p,⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ ⋃τ₁×⋃τ₂}
  shows IsContinuous(ProductTopology(τ₁,τ₂),ProductTopology(η₁,η₂),g)
proof -
  let τ = ProductTopology(τ₁,τ₂)
  let η = ProductTopology(η₁,η₂)
  let X₁ = ⋃τ₁
  let X₂ = ⋃τ₂
  let Y₁ = ⋃η₁
  let Y₂ = ⋃η₂
  let B = ProductCollection(η₁,η₂)
  from A1 A2 have τ {is a topology} and η {is a topology}
    using Top_1_4_T1 by auto
  moreover have g: X₁×X₂ → Y₁×Y₂
```

```
  proof -
    { fix p assume p ∈ X₁×X₂
      hence fst(p) ∈ X₁ and snd(p) ∈ X₂ by auto
      from A3a ‹fst(p) ∈ X₁› have f₁(fst(p)) ∈ Y₁
        by (rule apply_funtype)
      moreover from A3b ‹snd(p) ∈ X₂› have f₂(snd(p)) ∈ Y₂
        by (rule apply_funtype)
      ultimately have ⟨f₁(fst(p)),f₂(snd(p))⟩ ∈ ⋃η₁×⋃η₂ by auto
    } hence ∀p ∈ X₁×X₂. ⟨f₁(fst(p)),f₂(snd(p))⟩ ∈ Y₁×Y₂
      by simp
    with A5 show g: X₁×X₂ → Y₁×Y₂ using ZF_fun_from_total
      by simp
  qed
  moreover from A1 A2 have ⋃τ = X₁×X₂ and ⋃η = Y₁×Y₂
    using Top_1_4_T1 by auto
  ultimately have two_top_spaces0(τ,η,g) using two_top_spaces0_def
    by simp
  moreover from A2 have B {is a base for} η using Top_1_4_T1
    by simp
  moreover have ∀U∈B. g-(U) ∈ τ
  proof
    fix U assume U∈B
    then obtain V W where V ∈ η₁ W ∈ η₂ and U = V×W
      using ProductCollection_def by auto
    with A3a A3b A5 have g-(U) = f₁-(V) × f₂-(W)
      using cart_prod_fun_vimage by simp
    moreover from A1 A4 ‹V ∈ η₁› ‹W ∈ η₂› have f₁-(V) × f₂-(W) ∈ τ
      using IsContinuous_def prod_open_open_prod by simp
    ultimately show g-(U) ∈ τ by simp
  qed
  ultimately show thesis using two_top_spaces0.Top_ZF_2_1_L5
    by simp
qed
```

A reformulation of the cart_prod_cont lemma above in slightly different notation.

```
theorem (in two_top_spaces0) product_cont_functions:
  assumes f:X₁→X₂ g:⋃τ₃→⋃τ₄
    IsContinuous(τ₁,τ₂,f) IsContinuous(τ₃,τ₄,g)
    τ₄{is a topology} τ₃{is a topology}
  shows IsContinuous(ProductTopology(τ₁,τ₃),ProductTopology(τ₂,τ₄),{⟨⟨x,y⟩,⟨fx,gy⟩⟩.
⟨x,y⟩∈X₁×⋃τ₃})
proof -
  have {⟨⟨x,y⟩,⟨fx,gy⟩⟩. ⟨x,y⟩∈X₁×⋃τ₃} = {⟨p,⟨f(fst(p)),g(snd(p))⟩⟩. p
∈ X₁×⋃τ₃}
    by force
  with tau1_is_top tau2_is_top assms show thesis using cart_prod_cont
by simp
qed
```

A special case of `cart_prod_cont` when the function acting on the second axis is the identity.

```
lemma cart_prod_cont1:
 assumes A1: τ₁ {is a topology} and A1a: τ₂ {is a topology} and
  A2: η₁ {is a topology}  and
  A3: f₁:⋃τ₁→⋃η₁ and A4: IsContinuous(τ₁,η₁,f₁) and
  A5: g = {⟨p, ⟨f₁(fst(p)),snd(p)⟩⟩. p ∈ ⋃τ₁×⋃τ₂}
  shows IsContinuous(ProductTopology(τ₁,τ₂),ProductTopology(η₁,τ₂),g)
proof -
  let f₂ = id(⋃τ₂)
  have ∀x∈⋃τ₂. f₂(x) = x using id_conv by blast
  hence I: ∀p ∈ ⋃τ₁×⋃τ₂. snd(p) = f₂(snd(p)) by simp
  note A1 A1a A2 A1a A3
  moreover have f₂:⋃τ₂→⋃τ₂  using id_type by simp
  moreover note A4
  moreover have IsContinuous(τ₂,τ₂,f₂) using id_cont by simp
  moreover have g = {⟨p, ⟨f₁(fst(p)),f₂(snd(p))⟩ ⟩. p ∈ ⋃τ₁×⋃τ₂}
  proof
    from A5 I show  g ⊆ {⟨p, ⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ ⋃τ₁×⋃τ₂}
      by auto
    from A5 I show {⟨p, ⟨f₁(fst(p)),f₂(snd(p))⟩⟩. p ∈ ⋃τ₁×⋃τ₂} ⊆ g
      by auto
  qed
  ultimately show thesis by (rule cart_prod_cont)
qed
```

Having two continuous mappings f, g we can construct a third one with values in the cartesian product of the codomains of f, g, defined by $x \mapsto \langle f(x), g(x)\rangle$.

```
lemma (in prod_top_spaces0) cont_funcs_prod:
  assumes f:X₁→X₂ g:X₁→X₃ IsContinuous(τ₁,τ₂,f) IsContinuous(τ₁,τ₃,g)
  defines h ≡ {⟨x,⟨f(x),g(x)⟩⟩. x∈X₁}
  shows IsContinuous(τ₁,ProductTopology(τ₂,τ₃),h)
proof -
  let B = ProductCollection(τ₂,τ₃)
  have
    two_top_spaces0(τ₁,ProductTopology(τ₂,τ₃),h)
    B {is a base for} ProductTopology(τ₂,τ₃)
     ∀W∈B. h-(W) ∈ τ₁
  proof -
    from tau1_is_top tau2_is_top tau3_is_top assms(1,2,5)
      show two_top_spaces0(τ₁,ProductTopology(τ₂,τ₃),h)
        using vimage_prod Top_1_4_T1(1,3) unfolding two_top_spaces0_def

        by simp
    from tau2_is_top tau3_is_top showB {is a base for} ProductTopology(τ₂,τ₃)
      using Top_1_4_T1(2) by simp
    from tau1_is_top assms show ∀W∈B. h-(W) ∈ τ₁
      unfolding ProductCollection_def IsContinuous_def IsATopology_def
```

```
    using vimage_prod by simp
  qed
  then show thesis by (rule two_top_spaces0.Top_ZF_2_1_L5)
qed
```

Having two continuous mappings f, g we can construct a third one with values in the cartesian product of the codomains of f, g, defined by $x \mapsto \langle f(x), g(x)\rangle$. This is essentially the same as `cont_funcs_prod` but formulated in a way that is sometimes easier to apply. Recall that $\tau_2 \times_t \tau_3$ is a notation for the product topology of τ_1 and τ_2.

```
lemma cont_funcs_prod1:
  assumes τ₁ {is a topology} τ₂ {is a topology} τ₃ {is a topology} and
  {⟨x,p(x)⟩. x∈⋃τ₁} ∈ Cont(τ₁,τ₂) {⟨x,q(x)⟩. x∈⋃τ₁} ∈ Cont(τ₁,τ₃)
  shows {⟨x,⟨p(x),q(x)⟩⟩. x∈⋃τ₁} ∈ Cont(τ₁,τ₂×ₜτ₃)
proof -
  let X = ⋃τ₁
  let Y = ⋃τ₂
  let Z = ⋃τ₃
  let f = {⟨x,p(x)⟩. x∈X}
  let g = {⟨x,q(x)⟩. x∈X}
  let h = {⟨x,⟨p(x),q(x)⟩⟩. x∈X}
  from assms(4,5) have f:X→Y and g:X→Z unfolding Cont_def
    by auto
  with assms(2,3) have hFun: h:X→⋃(τ₂×ₜτ₃)
    using prod_fun_val(1) using Top_1_4_T1(3) by simp
  moreover have IsContinuous(τ₁,τ₂×ₜτ₃,h)
  proof -
    let B = ProductCollection(τ₂,τ₃)
    from assms(1,2,3) hFun have two_top_spaces0(τ₁,τ₂×ₜτ₃,h)
      using Top_1_4_T1(1) unfolding two_top_spaces0_def by simp
    moreover from assms(2,3) have B {is a base for} (τ₂×ₜτ₃)
      using Top_1_4_T1(2) by simp
    moreover have ∀W∈B. h-(W) ∈ τ₁
    proof -
      { fix W assume W∈B
        then obtain U V where U∈τ₂ V∈τ₃ W=U×V
          unfolding ProductCollection_def by auto
        have ∀x∈X. ⟨p(x),q(x)⟩ = ⟨f(x),g(x)⟩
        proof -
          { fix x assume x∈X
            then have ⟨p(x),q(x)⟩ = ⟨f(x),g(x)⟩
              using ZF_fun_from_tot_val1 by simp
          } thus thesis by auto
        qed
        then have h = {⟨x,⟨f(x),g(x)⟩⟩. x∈X} by (rule set_comp_eq)
        with assms(1,4,5) ‹f:X→Y› ‹g:X→Z› ‹U∈τ₂› ‹V∈τ₃› ‹W=U×V›
        have h-(W) ∈ τ₁ using vimage_prod(3)
          unfolding Cont_def IsContinuous_def IsATopology_def by auto
```

```
    } thus thesis by simp
  qed
  ultimately show thesis by (rule two_top_spaces0.Top_ZF_2_1_L5)
 qed
 ultimately show thesis unfolding Cont_def by simp
qed
```

Two continuous functions into a Hausdorff space are equal on a closed set. Note that in the lemma below f is assumed to map X_1 to X_2 in the locale, we only need to add a similar assumption for the second function.

```
lemma (in two_top_spaces0) two_fun_eq_closed:
  assumes g:X₁→X₂ f {is continuous} g {is continuous}  τ₂ {is T₂}
  shows {x∈X₁. f(x)=g(x)} {is closed in} τ₁
proof -
  let D = {x∈X₁. f(x)=g(x)}
  let h = {⟨x,⟨f(x),g(x)⟩⟩. x∈X₁}
  have h-({⟨y,y⟩. y∈X₂}) {is closed in} τ₁
    proof -
      have two_top_spaces0(τ₁,ProductTopology(τ₂,τ₂),h)
      proof
        from tau1_is_top tau2_is_top
        show τ₁ {is a topology} and ProductTopology(τ₂,τ₂) {is a topology}
          using Top_1_4_T1(1) by auto
        from tau1_is_top tau2_is_top fmapAssum assms(1)
        show h : ⋃τ₁ → ⋃ProductTopology(τ₂, τ₂)
          using vimage_prod(1) Top_1_4_T1(3) by simp
      qed
      moreover
      have IsContinuous(τ₁,ProductTopology(τ₂,τ₂),h)
      proof -
        from tau1_is_top tau2_is_top have prod_top_spaces0(τ₁,τ₂,τ₂)
          unfolding prod_top_spaces0_def by simp
        with fmapAssum assms(1,2,3) show thesis
          using prod_top_spaces0.cont_funcs_prod by simp
      qed
      moreover
      from tau2_is_top assms(4)
        have {⟨y,y⟩. y∈X₂} {is closed in} ProductTopology(τ₂,τ₂)
          using t2_iff_diag_closed by simp
      ultimately show thesis using two_top_spaces0.TopZF_2_1_L1
        by simp
    qed
  with fmapAssum assms(1) show thesis using vimage_prod(4)
    by simp
qed
```

Closure of an image of a singleton by a relation in $X \times Y$ is contained in the image of this singleton by the closure of the relation (in the product topology). Compare the proof of Metamath's theorem with the same name.

```
lemma imasncls:
  assumes T {is a topology}  S {is a topology} R ⊆ (⋃T)×(⋃S) x∈⋃T
  shows Closure(R{x},S) ⊆ Closure(R,T×tS){x}
proof -
  let X = ⋃T
  let Y = ⋃S
  let f = {⟨y,⟨x,y⟩⟩. y∈Y}
  from assms(3) have R{x} = f-(R) by blast
  hence Closure(R{x},S) = Closure(f-(R),S) by simp
  also have Closure(f-(R),S) ⊆ f-(Closure(R,T×tS))
  proof -
    from assms(1,2,4) have f ∈ Cont(S,T×tS)
      using const_cont_sp id_cont_sp cont_funcs_prod1 by simp
    with assms(1,2,3) have
      two_top_spaces0(S,T×tS,f) IsContinuous(S,T×tS,f) R ⊆ ⋃(T×tS)
      unfolding Cont_def two_top_spaces0_def using Top_1_4_T1(1,3)
      by auto
    then show Closure(f-(R),S) ⊆ f-(Closure(R,T×tS))
      using two_top_spaces0.im_cl_in_cl_im by simp
  qed
  also
  have Closure(R,T×tS) ⊆ X×Y
  proof -
    from assms(1,2,3) have topology0(T×tS) R ⊆ ⋃(T×tS)
      unfolding topology0_def using Top_1_4_T1(1,3) by auto
    then have Closure(R,T×tS) ⊆ ⋃(T×tS) by (rule topology0.Top_3_L11)
    with assms(1,2) show thesis using Top_1_4_T1(3) by simp
  qed
  with assms(4) have f-(Closure(R,T×tS)) = Closure(R,T×tS){x} by blast
  finally show Closure(R{x},S) ⊆ Closure(R,T×tS){x} by simp
qed
```

66.6 Pasting lemma

The classical pasting lemma states that if U_1, U_2 are both open (or closed) and a function is continuous when restricted to both U_1 and U_2 then it is continuous when restricted to $U_1 \cup U_2$. In this section we prove a generalization statement stating that the set $\{U \in \tau_1 | f|_U$ is continuous $\}$ is a topology.

A typical statement of the pasting lemma uses the notion of a function restricted to a set being continuous without specifying the topologies with respect to which this continuity holds. In `two_top_spaces0` context the notation `g {is continuous}` means continuity wth respect to topologies τ_1, τ_2. The next lemma is a special case of `partial_fun_cont` and states that if for some set $A \subseteq X_1 = \bigcup \tau_1$ the function $f|_A$ is continuous (with respect to (τ_1, τ_2)), then A has to be open. This clears up terminology and indicates why we need to pay attention to the issue of which topologies we talk about when we say that the restricted (to some closed set for example) function is continuos.

```
lemma (in two_top_spaces0) restriction_continuous1:
  assumes A1: A ⊆ X₁ and A2: restrict(f,A) {is continuous}
  shows A ∈ τ₁
proof -
  from assms have two_top_spaces1(τ₁,τ₂) and
    restrict(f,A):A→X₂ and restrict(f,A) {is continuous}
    using tau1_is_top tau2_is_top two_top_spaces1_def fmapAssum restrict_fun
      by auto
  then show thesis using two_top_spaces1.partial_fun_cont by simp
qed
```

If a fuction is continuous on each set of a collection of open sets, then it is continuous on the union of them. We could use continuity with respect to the relative topology here, but we know that on open sets this is the same as the original topology.

```
lemma (in two_top_spaces0) pasting_lemma1:
  assumes A1: M ⊆ τ₁ and A2: ∀U∈M. restrict(f,U)  {is continuous}
  shows restrict(f,⋃M) {is continuous}
proof -
  { fix V assume V∈τ₂
    from A1 have ⋃M ⊆ X₁ by auto
    then have restrict(f,⋃M)-(V) = f-(V) ∩ (⋃M)
      using func1_2_L1 fmapAssum by simp
    also have ... = ⋃ {f-(V) ∩ U. U∈M} by auto
    finally have restrict(f,⋃M)-(V) = ⋃ {f-(V) ∩ U. U∈M} by simp
    moreover
    have {f-(V) ∩ U. U∈M} ∈ Pow(τ₁)
    proof -
      { fix W assume W ∈ {f-(V) ∩ U. U∈M}
        then obtain U where U∈M and I: W = f-(V) ∩ U by auto
        with A2 have restrict(f,U) {is continuous} by simp
        with ‹V∈τ₂› have restrict(f,U)-(V) ∈ τ₁
          using IsContinuous_def by simp
        moreover from ‹⋃M ⊆ X₁› and ‹U∈M›
        have restrict(f,U)-(V) = f-(V) ∩ U
          using fmapAssum func1_2_L1 by blast
        ultimately have f-(V) ∩ U ∈ τ₁ by simp
        with I have W ∈ τ₁ by simp
      } then show thesis by auto
    qed
    then have  ⋃{f-(V) ∩ U. U∈M} ∈ τ₁
       using tau1_is_top IsATopology_def by auto
    ultimately have restrict(f,⋃M)-(V) ∈ τ₁
      by simp
  } then show thesis using IsContinuous_def by simp
qed
```

If a function is continuous on two sets, then it is continuous on intersection.

```
lemma (in two_top_spaces0) cont_inter_cont:
  assumes A1: A ⊆ X₁ B ⊆ X₁ and
  A2: restrict(f,A)  {is continuous}  restrict(f,B)  {is continuous}
  shows restrict(f,A∩B)  {is continuous}
proof -
  { fix V assume V∈τ₂
    with assms have
      restrict(f,A)-(V) = f-(V) ∩ A  restrict(f,B)-(V) = f-(V) ∩ B and
      restrict(f,A)-(V) ∈ τ₁ and restrict(f,B)-(V) ∈ τ₁
        using func1_2_L1 fmapAssum IsContinuous_def by auto
    then have (restrict(f,A)-(V)) ∩ (restrict(f,B)-(V)) = f-(V) ∩ (A∩B)
      by auto
    moreover
    from A2 ‹V∈τ₂› have
      restrict(f,A)-(V) ∈ τ₁ and restrict(f,B)-(V) ∈ τ₁
      using IsContinuous_def by auto
    then have (restrict(f,A)-(V)) ∩ (restrict(f,B)-(V)) ∈ τ₁
      using tau1_is_top IsATopology_def by simp
    moreover
    from A1 have (A∩B) ⊆ X₁ by auto
    then have restrict(f,A∩B)-(V) = f-(V) ∩ (A∩B)
      using func1_2_L1 fmapAssum by simp
  ultimately have restrict(f,A∩B)-(V) ∈ τ₁ by simp
  } then show thesis using  IsContinuous_def by auto
qed
```

The collection of open sets U such that f restricted to U is continuous, is a topology.

```
theorem (in two_top_spaces0) pasting_theorem:
  shows {U ∈ τ₁. restrict(f,U) {is continuous}} {is a topology}
proof -
  let T = {U ∈ τ₁. restrict(f,U) {is continuous}}
  have ∀M∈Pow(T). ⋃M ∈ T
  proof
    fix M assume M ∈ Pow(T)
    then have restrict(f,⋃M) {is continuous}
      using pasting_lemma1 by auto
    with ‹M ∈ Pow(T)› show ⋃M ∈ T
      using tau1_is_top IsATopology_def by auto
  qed
  moreover have ∀U∈T.∀V∈T. U∩V ∈ T
    using cont_inter_cont tau1_is_top IsATopology_def by auto
  ultimately show thesis using IsATopology_def by simp
qed
```

0 is continuous.

```
corollary (in two_top_spaces0) zero_continuous: shows 0 {is continuous}
```

```
proof -
  let T = {U ∈ τ₁. restrict(f,U) {is continuous}}
  have T {is a topology} by (rule pasting_theorem)
  then have 0∈T by (rule empty_open)
  hence restrict(f,0) {is continuous} by simp
  moreover have restrict(f,0) = 0 by simp
  ultimately show thesis by simp
qed
```

```
end
```

67 Topology 4

```
theory Topology_ZF_4 imports Topology_ZF_1 Order_ZF func1 NatOrder_ZF
begin
```

This theory deals with convergence in topological spaces. Contributed by Daniel de la Concepcion.

67.1 Nets

Nets are a generalization of sequences. It is known that sequences do not determine the behavior of the topological spaces that are not first countable; i.e., have a countable neighborhood base for each point. To solve this problem, nets were defined so that the behavior of any topological space can be thought in terms of convergence of nets.

We say that a relation r `directs` a set X if the relation is reflexive, transitive on X and for every two elements x, y of X there is some element z such that both x and y are in the relation with z. Note that this naming is a bit inconsistent with what is defined in `Order_ZF` where we define what it means that r `up-directs` X (the third condition in the definition below) or r `down-directs` X. This naming inconsistency will be fixed in the future (maybe).

```
definition
  IsDirectedSet (_ directs _ 90)
  where r directs X ≡ refl(X,r) ∧ trans(r) ∧ (∀x∈X. ∀y∈X. ∃z∈X. ⟨x,z⟩∈r
∧ ⟨y,z⟩∈r)
```

Any linear order is a directed set; in particular $(\mathbb{N}, \leq)$.

```
lemma linorder_imp_directed:
  assumes IsLinOrder(X,r)
  shows r directs X
proof-
  from assms have trans(r) using IsLinOrder_def by auto
```

```
  moreover
  from assms have r:refl(X,r) using IsLinOrder_def total_is_refl by auto
  moreover
  {
    fix x y
    assume R: x∈X y∈X
    with assms have ⟨x,y⟩∈r ∨ ⟨y,x⟩∈r using IsLinOrder_def IsTotal_def
by auto
    with r have (⟨x,y⟩∈r ∧ ⟨y,y⟩∈r)∨(⟨y,x⟩∈r ∧ ⟨x,x⟩∈r) using R refl_def
by auto
    then have ∃z∈X. ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r using R by auto
  }
  ultimately show thesis using IsDirectedSet_def function_def by auto
qed
```

Natural numbers are a directed set.

```
corollary Le_directs_nat:
  shows IsLinOrder(nat,Le) Le directs nat
proof -
  show IsLinOrder(nat,Le)  by (rule NatOrder_ZF_1_L2)
  then show Le directs nat using linorder_imp_directed by auto
qed
```

We are able to define the concept of net, now that we now what a directed set is.

```
definition
  IsNet (_ {is a net on} _ 90)
  where N {is a net on} X ≡ fst(N):domain(fst(N))→X ∧ (snd(N) directs
domain(fst(N))) ∧ domain(fst(N))≠0
```

Provided a topology and a net directed on its underlying set, we can talk about convergence of the net in the topology.

```
definition (in topology0)
  NetConverges (_ →N _ 90)
  where N {is a net on} ⋃T ⟹ N →N x ≡
  (x∈⋃T) ∧ (∀U∈Pow(⋃T). (x∈int(U) ⟶ (∃t∈domain(fst(N)). ∀m∈domain(fst(N)).

    (⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U))))
```

One of the most important directed sets, is the neighborhoods of a point.

```
theorem (in topology0) directedset_neighborhoods:
  assumes x∈⋃T
  defines Neigh≡{U∈Pow(⋃T). x∈int(U)}
  defines r≡{⟨U,V⟩∈(Neigh × Neigh). V⊆U}
  shows r directs Neigh
proof-
  {
    fix U
```

```
    assume U ∈ Neigh
    then have ⟨U,U⟩ ∈ r using r_def by auto
  }
  then have refl(Neigh,r) using refl_def by auto
  moreover
  {
    fix U V W
    assume ⟨U,V⟩ ∈ r ⟨V,W⟩ ∈ r
    then have U ∈ Neigh W ∈ Neigh W⊆U using r_def by auto
    then have ⟨U,W⟩∈r using r_def by auto
  }
  then have trans(r) using trans_def by blast
  moreover
  {
    fix A B
    assume p: A∈Neigh B∈Neigh
    have A∩B ∈ Neigh
    proof-
      from p have A∩B ∈ Pow(⋃T) using Neigh_def by auto
      moreover
      { from p have x∈int(A)x∈int(B) using Neigh_def by auto
        then have x∈int(A)∩int(B) by auto
        moreover
        { have int(A)∩int(B)⊆A∩B using Top_2_L1  by auto
          moreover have int(A)∩int(B)∈T
            using Top_2_L2 Top_2_L2 topSpaceAssum IsATopology_def by blast
          ultimately have int(A)∩int(B)⊆int(A∩B)
          using Top_2_L5 by auto
        }
        ultimately have x ∈ int(A∩B) by auto
      }
      ultimately show thesis using Neigh_def by auto
    qed
    moreover from ‹A∩B ∈ Neigh› have ⟨A,A∩B⟩∈r ∧ ⟨B,A∩B⟩∈r
      using r_def p by auto
    ultimately
    have ∃z∈Neigh. ⟨A,z⟩∈r ∧ ⟨B,z⟩∈r by auto
  }
  ultimately show thesis using IsDirectedSet_def by auto
qed
```

There can be nets directed by the neighborhoods that converge to the point; if there is a choice function.

```
theorem (in topology0) net_direct_neigh_converg:
  assumes x∈⋃T
  defines Neigh≡{U∈Pow(⋃T). x∈int(U)}
  defines r≡{⟨U,V⟩∈(Neigh × Neigh). V⊆U}
  assumes f:Neigh→⋃T ∀U∈Neigh. f(U) ∈ U
  shows ⟨f,r⟩ →N x
```

```
proof -
  from assms(4) have dom_def: Neigh = domain(f) using Pi_def by auto
  moreover
    have ⋃T∈T using topSpaceAssum IsATopology_def by auto
    then have int(⋃T)=⋃T using Top_2_L3 by auto
    with assms(1) have ⋃T∈Neigh using Neigh_def by auto
    then have ⋃T∈domain(fst(⟨f,r⟩)) using dom_def by auto
  moreover from assms(4) dom_def have fst(⟨f,r⟩):domain(fst(⟨f,r⟩))→⋃T

    by auto
  moreover from assms(1,2,3) dom_def have snd(⟨f,r⟩) directs domain(fst(⟨f,r⟩))

      using directedset_neighborhoods by simp
  ultimately have Net: ⟨f,r⟩ {is a net on} ⋃T unfolding IsNet_def by
auto
  {
    fix U
    assume U ∈ Pow(⋃T) x ∈ int(U)
    then have U ∈ Neigh using Neigh_def by auto
    then have t: U ∈ domain(f) using dom_def by auto
    {
      fix W
      assume A: W∈domain(f) ⟨U,W⟩∈r
      then have W∈Neigh using dom_def by auto
      with assms(5) have fW∈W by auto
      with A(2) r_def have fW∈U by auto
    }
    then have ∀W∈domain(f). (⟨U,W⟩∈r ⟶ fW∈U) by auto
    with t have ∃V∈domain(f). ∀W∈domain(f). (⟨V,W⟩∈r ⟶ fW∈U) by auto
  }
  then have ∀U∈Pow(⋃T). (x∈int(U) ⟶ (∃V∈domain(f). ∀W∈domain(f).
(⟨V,W⟩∈r ⟶ f(W) ∈ U)))
    by auto
  with assms(1) Net show thesis using NetConverges_def by auto
qed
```

67.2 Filters

Nets are a generalization of sequences that can make us see that not all topological spaces can be described by sequences. Nevertheless, nets are not always the tool used to deal with convergence. The reason is that they make use of directed sets which are completely unrelated with the topology.

The topological tools to deal with convergence are what is called filters.

```
definition
  IsFilter (_ {is a filter on} _ 90)
  where 𝔉 {is a filter on} X ≡ (∅∉𝔉) ∧ (X∈𝔉) ∧ 𝔉⊆Pow(X) ∧
  (∀A∈𝔉. ∀B∈𝔉. A∩B∈𝔉) ∧ (∀B∈𝔉. ∀C∈Pow(X). B⊆C ⟶ C∈𝔉)
```

The next lemma splits the the definition of a filter into four conditions to make it easier to reference each one separately in proofs.

```
lemma is_filter_def_split: assumes 𝔉 {is a filter on} X
  shows ∅∉𝔉 X∈𝔉 𝔉⊆Pow(X)
    ∀A∈𝔉. ∀B∈𝔉. A∩B∈𝔉 and ∀B∈𝔉. ∀C∈Pow(X). B⊆C ⟶ C∈𝔉
  using assms unfolding IsFilter_def by auto
```

Filters are closed with respect to taking finite intersections.

```
lemma filter_fin_inter_closed:
  assumes 𝔉 {is a filter on} X M∈FinPow(𝔉)\{∅}
  shows ⋂M ∈ 𝔉
  using assms is_filter_def_split(4) inter_two_inter_fin by simp
```

Filters are closed with respect to taking supersets.

```
lemma filter_superset_closed:
  assumes 𝔉 {is a filter on} X 𝒜⊆𝔉
  shows Supersets(X,𝒜) ⊆ 𝔉
  using assms is_filter_def_split(5) unfolding Supersets_def
  by auto
```

Not all the sets of a filter are needed to be consider at all times; as it happens with a topology we can consider bases.

```
definition
  IsBaseFilter (_ {is a base filter} _ 90)
  where C {is a base filter} 𝔉 ≡ C⊆𝔉 ∧ 𝔉={A∈Pow(⋃𝔉). (∃D∈C. D⊆A)}
```

Not every set is a base for a filter, as it happens with topologies, there is a condition to be satisfied.

```
definition
  SatisfiesFilterBase (_ {satisfies the filter base condition} 90)
  where C {satisfies the filter base condition} ≡ (∀A∈C. ∀B∈C. ∃D∈C.
D⊆A∩B) ∧ C≠0 ∧ 0∉C
```

Every set of a filter contains a set from the filter's base.

```
lemma basic_element_filter:
  assumes A∈𝔉 and C {is a base filter} 𝔉
  shows ∃D∈C. D⊆A
proof-
  from assms(2) have t:𝔉={A∈Pow(⋃𝔉). (∃D∈C. D⊆A)} using IsBaseFilter_def
by auto
  with assms(1) have A∈{A∈Pow(⋃𝔉). (∃D∈C. D⊆A)} by auto
  then have A∈Pow(⋃𝔉) ∃D∈C. D⊆A by auto
  then show thesis by auto
qed
```

The following two results state that the filter base condition is necessary and sufficient for the filter generated by a base, to be an actual filter. The third result, rewrites the previous two.

```
theorem basic_filter_1:
  assumes C {is a base filter} 𝔉 and C {satisfies the filter base condition}
  shows 𝔉 {is a filter on} ⋃𝔉
proof-
  {
    fix A B
    assume AF: A∈𝔉 and BF: B∈𝔉
    with assms(1) have ∃DA∈C. DA⊆A using  basic_element_filter by simp
    then obtain DA where perA: DA∈C and subA: DA⊆A by auto
    from BF assms have ∃DB∈C. DB⊆B using  basic_element_filter by simp
    then obtain DB where perB: DB∈C and subB: DB⊆B by auto
    from assms(2) perA perB have ∃D∈C. D⊆DA∩DB
      unfolding SatisfiesFilterBase_def by auto
    then obtain D where D∈C D⊆DA∩DB by auto
    with subA subB AF BF have A∩B∈{A ∈ Pow(⋃𝔉) . ∃D∈C. D ⊆ A} by auto
    with assms(1) have A∩B∈𝔉 unfolding IsBaseFilter_def by auto
  }
  moreover
  {
    fix A B
    assume AF: A∈𝔉 and BS: B∈Pow(⋃𝔉) and sub: A⊆B
    from assms(1) AF have ∃D∈C. D⊆A using basic_element_filter by auto
    then obtain D where D⊆A D∈C by auto
    with sub BS have B∈{A∈Pow(⋃𝔉). ∃D∈C. D⊆A} by auto
    with assms(1) have B∈𝔉 unfolding IsBaseFilter_def by auto
    }
  moreover
  from assms(2) have C≠0 using SatisfiesFilterBase_def by auto
  then obtain D where D∈C by auto
  with assms(1) have D⊆⋃𝔉 using IsBaseFilter_def by auto
  with ‹D∈C› have ⋃𝔉∈{A∈Pow(⋃𝔉). ∃D∈C. D⊆A} by auto
  with assms(1) have ⋃𝔉∈𝔉 unfolding IsBaseFilter_def by auto
  moreover
  {
    assume 0∈𝔉
    with assms(1) have ∃D∈C. D⊆0 using basic_element_filter by simp

    then obtain D where D∈CD⊆0 by auto
    then have D∈C D=0 by auto
    with assms(2) have False using SatisfiesFilterBase_def by auto
  }
  then have 0∉𝔉 by auto
  ultimately show thesis using IsFilter_def by auto
qed
```

A base filter satisfies the filter base condition.

```
theorem basic_filter_2:
  assumes C {is a base filter} 𝔉 and 𝔉 {is a filter on} ⋃𝔉
  shows C {satisfies the filter base condition}
```

```
proof-
  {
    fix A B
    assume AF: A∈C and BF: B∈C
    then have A∈𝔉 and B∈𝔉 using assms(1) IsBaseFilter_def by auto
    then have A∩B∈𝔉 using assms(2) IsFilter_def by auto
    then have ∃D∈C. D⊆A∩B using assms(1) basic_element_filter by blast
  }
  then have ∀A∈C. ∀B∈C. ∃D∈C. D⊆A∩B by auto
  moreover
  {
    assume 0∈C
    then have 0∈𝔉 using assms(1) IsBaseFilter_def by auto
    then have False using assms(2) IsFilter_def by auto
  }
  then have 0∉C by auto
  moreover
  {
    assume C=0
    then have 𝔉=0 using assms(1) IsBaseFilter_def by auto
    then have False using assms(2) IsFilter_def by auto
  }
  then have C≠0 by auto
  ultimately show thesis using SatisfiesFilterBase_def by auto
qed
```

A base filter for a collection satisfies the filter base condition iff that collection is in fact a filter.

```
theorem basic_filter:
  assumes C {is a base filter} 𝔉
  shows (C {satisfies the filter base condition}) ⟷ (𝔉 {is a filter
on} ⋃𝔉)
using assms basic_filter_1 basic_filter_2 by auto
```

A base for a filter determines a filter up to the underlying set.

```
theorem base_unique_filter:
  assumes C {is a base filter} 𝔉1and C {is a base filter} 𝔉2
  shows 𝔉1=𝔉2 ⟷ ⋃𝔉1=⋃𝔉2
using assms unfolding IsBaseFilter_def by auto
```

Suppose that we take any nonempty collection C of subsets of some set X. Then this collection is a base filter for the collection of all supersets (in X) of sets from C.

```
theorem base_unique_filter_set1:
  assumes C ⊆ Pow(X) and C≠0
  shows C {is a base filter} {A∈Pow(X). ∃D∈C. D⊆A} and ⋃{A∈Pow(X).
∃D∈C. D⊆A}=X
proof-
```

```
  from assms(1) have C⊆{A∈Pow(X). ∃D∈C. D⊆A} by auto
  moreover
  from assms(2) obtain D where D∈C by auto
  then have D⊆X using assms(1) by auto
  with ‹D∈C› have X∈{A∈Pow(X). ∃D∈C. D⊆A} by auto
  then show ⋃{A∈Pow(X). ∃D∈C. D⊆A}=X by auto
  ultimately
  show C {is a base filter} {A∈Pow(X). ∃D∈C. D⊆A} using IsBaseFilter_def
by auto
qed
```

A collection C that satisfies the filter base condition is a base filter for some other collection $\mathfrak{F}$ iff $\mathfrak{F}$ is the collection of supersets of C.

```
theorem base_unique_filter_set2:
  assumes C⊆Pow(X) and C {satisfies the filter base condition}
  shows ((C {is a base filter} 𝔉) ∧ ⋃𝔉=X) ⟷ 𝔉={A∈Pow(X). ∃D∈C. D⊆A}
  using assms IsBaseFilter_def SatisfiesFilterBase_def base_unique_filter_set1
   by auto
```

A simple corollary from the previous lemma.

```
corollary base_unique_filter_set3:
  assumes C⊆Pow(X) and C {satisfies the filter base condition}
  shows C {is a base filter} {A∈Pow(X). ∃D∈C. D⊆A} and ⋃{A∈Pow(X).
∃D∈C. D⊆A} = X
proof -
  let 𝔉 = {A∈Pow(X). ∃D∈C. D⊆A}
  from assms have (C {is a base filter} 𝔉) ∧ ⋃𝔉=X
    using base_unique_filter_set2 by simp
  thus C {is a base filter} 𝔉 and ⋃𝔉 = X
    by auto
qed
```

Every filter can be expanded by a set

```
lemma extend_filter:
  assumes A∈Pow(X) 𝔉 {is a filter on} X A≠0 A∉𝔉 X\A∉𝔉
  shows ∃𝔊. (𝔊 {is a filter on} X) ∧ A∈𝔊 ∧ 𝔉⊆𝔊
proof
  from assms(2,3) have ne: {F∩A. F∈𝔉}≠∅ unfolding IsFilter_def
    by auto
  moreover
  { assume ∅ ∈ {F∩A. F∈𝔉}
    then obtain F where F: F∈𝔉 F∩A = ∅ by auto
    with assms(2) have f: F ⊆ X\A unfolding IsFilter_def by auto
    from assms(2) F(1) have ∀C∈Pow(X). F ⊆ C ⟶ C ∈ 𝔉
      unfolding IsFilter_def by auto
    with assms(5) ‹F ⊆ X\A› have False by auto
  } hence ∅ ∉ {F∩A. F∈𝔉} by auto
  moreover
  { fix x y assume as: x ∈ {F∩A. F∈𝔉} y ∈ {F∩A. F∈𝔉}
```

```
    then obtain fₓ f_y where ff: x=fₓ∩A y=f_y∩A fₓ∈𝔉 f_y∈𝔉
      by auto
    hence x∩y = fₓ∩f_y∩A by auto
    with assms(2) ff have ∃D∈{F∩A. F∈𝔉}. D⊆x∩y
      unfolding IsFilter_def by blast
  } hence ∀A_a∈{F∩A. F∈𝔉}. ∀B∈{F∩A. F∈𝔉}. ∃D∈{F∩A. F∈𝔉}. D⊆A_a∩B
    by auto
  ultimately have
    baseCond: {F∩A. F∈𝔉} {satisfies the filter base condition}
    unfolding SatisfiesFilterBase_def by blast
  let F = {A_A∈Pow(X). (∃D∈{F∩A. F∈𝔉}. D⊆A_A)}
  have rule: {F∩A. F∈𝔉} ⊆ Pow(X) ∧  {F∩A. F∈𝔉} ≠ ∅ ⟶
    ({F∩A. F∈𝔉} {is a base filter} {A_A∈Pow(X). (∃D∈{F∩A. F∈𝔉}. D⊆A_A)})
    using base_unique_filter_set1(1) by blast
  from assms(2) have p: {F∩A. F∈𝔉} ⊆ Pow(X) unfolding IsFilter_def
    by auto
  with ne rule have base: {F∩A. F∈𝔉} {is a base filter} F by auto
  have {F∩A. F∈𝔉}⊆Pow(X) ∧ ({F∩A. F∈𝔉} {satisfies the filter base condition})
      ⟶ ⋃F = X
    using base_unique_filter_set3(2) by blast
  with p base baseCond have F {is a filter on} X using basic_filter by
auto
  moreover from assms(1,2) have A∈F and 𝔉⊆F unfolding IsFilter_def

    by auto
  ultimately show (F {is a filter on} X) ∧ A ∈ F ∧ 𝔉 ⊆ F by auto
qed
```

The convergence for filters is much easier concept to write. Given a topology and a filter on the same underlying set, we can define convergence as containing all the neighborhoods of the point.

```
definition (in topology0)
  FilterConverges (_ →_F _ 50) where
  𝔉{is a filter on}⋃T  ⟹ 𝔉→_Fx ≡
  x∈⋃T ∧ ({U∈Pow(⋃T). x∈int(U)} ⊆ 𝔉)
```

The neighborhoods of a point form a filter that converges to that point.

```
lemma (in topology0) neigh_filter:
  assumes x∈⋃T
  defines Neigh≡{U∈Pow(⋃T). x∈int(U)}
  shows Neigh {is a filter on}⋃T and Neigh →_F x
proof-
  {
    fix A B
    assume p:A∈Neigh B∈Neigh
    have A∩B∈Neigh
    proof-
      from p have A∩B∈Pow(⋃T) using Neigh_def by auto
      moreover
```

```
      {from p have x∈int(A) x∈int(B) using Neigh_def by auto
      then have x∈int(A)∩int(B) by auto
      moreover
      { have int(A)∩int(B)⊆A∩B using Top_2_L1 by auto
        moreover have int(A)∩int(B)∈T
          using Top_2_L2 topSpaceAssum IsATopology_def by blast
        ultimately have int(A)∩int(B)⊆int(A∩B) using Top_2_L5 by auto}
        ultimately have x∈int(A∩B) by auto
      }
      ultimately show thesis using Neigh_def by auto
    qed
    }
  moreover
  {
    fix A B
    assume A: A∈Neigh and B: B∈Pow(⋃T) and sub: A⊆B
    from sub have int(A)∈T int(A)⊆B using Top_2_L2 Top_2_L1
      by auto
    then have int(A)⊆int(B) using Top_2_L5  by auto
    with A have x∈int(B) using Neigh_def by auto
    with B have B∈Neigh using Neigh_def by auto
    }
  moreover
  {
    assume 0∈Neigh
    then have x∈Interior(0,T) using Neigh_def by auto
    then have x∈0 using Top_2_L1 by auto
    then have False by auto
    }
  then have 0∉Neigh by auto
  moreover
  have ⋃T∈T using topSpaceAssum IsATopology_def by auto
  then have Interior(⋃T,T)=⋃T using Top_2_L3 by auto
  with assms(1) have ab: ⋃T∈Neigh unfolding Neigh_def by auto
  moreover have Neigh⊆Pow(⋃T) using Neigh_def by auto
  ultimately show Neigh {is a filter on} ⋃T using IsFilter_def
    by auto
  moreover from ab have ⋃Neigh=⋃T unfolding Neigh_def by auto
  ultimately show Neigh →F x using FilterConverges_def assms(1) Neigh_def
by auto
qed
```

Note that with the net we built in a previous result, it wasn't clear that we could construct an actual net that converged to the given point without the axiom of choice. With filters, there is no problem.

Another positive point of filters is due to the existence of filter basis. If we have a basis for a filter, then the filter converges to a point iff every neighborhood of that point contains a basic filter element.

```
theorem (in topology0) convergence_filter_base1:
  assumes 𝔉 {is a filter on} ⋃T and C {is a base filter} 𝔉 and 𝔉 →F
x
  shows ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U) and x∈⋃T
proof -
  { fix U
    assume U⊆(⋃T) and x∈int(U)
    with assms(1,3) have U∈𝔉 using FilterConverges_def by auto
    with assms(2) have ∃D∈C. D⊆U using basic_element_filter by blast
  } thus ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U) by auto
  from assms(1,3) show x∈⋃T using  FilterConverges_def by auto
qed
```

A sufficient condition for a filter to converge to a point.

```
theorem (in topology0) convergence_filter_base2:
  assumes 𝔉 {is a filter on} ⋃T and C {is a base filter} 𝔉
    and ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U) and x∈⋃T
  shows 𝔉 →F x
proof-
  {
    fix U
    assume AS: U∈Pow(⋃T) x∈int(U)
    then obtain D where pD:D∈C and s:D⊆U using assms(3) by blast
    with assms(2) AS have D∈𝔉 and D⊆U and U∈Pow(⋃T)
      using IsBaseFilter_def by auto
    with assms(1) have U∈𝔉 using IsFilter_def by auto
  }
  with assms(1,4) show thesis using FilterConverges_def by auto
qed
```

A necessary and sufficient condition for a filter to converge to a point.

```
theorem (in topology0) convergence_filter_base_eq:
  assumes 𝔉 {is a filter on} ⋃T and C {is a base filter} 𝔉
  shows (𝔉 →F x) ⟷ ((∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈⋃T)
proof
  assume 𝔉 →F x
  with assms show ((∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈⋃T)
    using convergence_filter_base1 by simp
  next
  assume (∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈⋃T
  with assms show 𝔉 →F x using convergence_filter_base2
    by auto
qed
```

67.3 Relation between nets and filters

In this section we show that filters do not generalize nets, but still nets and filter are in w way equivalent as far as convergence is considered.

Let's build now a net from a filter, such that both converge to the same points.

```
definition
  NetOfFilter (Net(_) 40) where
  𝔉 {is a filter on} ⋃𝔉 ⟹ Net(𝔉) ≡
    ⟨{⟨A,fst(A)⟩. A∈{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F}},{⟨A,B⟩∈{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F}×{⟨x,F⟩∈(⋃𝔉)×𝔉.
x∈F}. snd(B)⊆snd(A)}⟩
```

Net of a filter is indeed a net.

```
theorem net_of_filter_is_net:
  assumes 𝔉 {is a filter on} X
  shows (Net(𝔉)) {is a net on} X
proof-
  from assms have X∈𝔉 𝔉⊆Pow(X) using IsFilter_def by auto
  then have uu:⋃𝔉=X by blast
  let f = {⟨A,fst(A)⟩. A∈{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F}}
  let r = {⟨A,B⟩∈{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F}×{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F}. snd(B)⊆snd(A)}
  have function(f) using function_def by auto
  moreover have relation(f) using relation_def by auto
  ultimately  have f:domain(f)→range(f) using function_imp_Pi
    by auto
  have dom:domain(f)={⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F} by auto
  have range(f)⊆⋃𝔉 by auto
  with ‹f:domain(f)→range(f)› have f:domain(f)→⋃𝔉 using fun_weaken_type
by auto
  moreover
  {
    {
      fix t
      assume pp:t∈domain(f)
      then have snd(t)⊆snd(t) by auto
      with dom pp have ⟨t,t⟩∈r by auto
    }
    then have refl(domain(f),r) using refl_def by auto
    moreover
    {
      fix t1 t2 t3
      assume ⟨t1,t2⟩∈r ⟨t2,t3⟩∈r
      then have snd(t3)⊆snd(t1) t1∈domain(f) t3∈domain(f) using dom
by auto
      then have ⟨t1,t3⟩∈r by auto
    }
    then have trans(r) using trans_def by auto
    moreover
    {
      fix x y
      assume as:x∈domain(f)y∈domain(f)
      then have snd(x)∈𝔉 snd(y)∈𝔉 by auto
      then have p:snd(x)∩snd(y)∈𝔉 using assms IsFilter_def by auto
```

```
      {
        assume snd(x)∩snd(y)=0
        with p have 0∈𝔉 by auto
        then have False using assms IsFilter_def by auto
      }
      then have snd(x)∩snd(y)≠0 by auto
      then obtain xy where xy∈snd(x)∩snd(y) by auto
      then have xy∈snd(x)∩snd(y) ⟨xy,snd(x)∩snd(y)⟩∈(⋃𝔉)×𝔉 using p
by auto
      then have ⟨xy,snd(x)∩snd(y)⟩∈{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F} by auto
      with dom have d:⟨xy,snd(x)∩snd(y)⟩∈domain(f) by auto
      with as have ⟨x,⟨xy,snd(x)∩snd(y)⟩⟩∈r ∧ ⟨y,⟨xy,snd(x)∩snd(y)⟩⟩∈r
by auto
      with d have ∃z∈domain(f). ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r  by blast
    }
    then have ∀x∈domain(f). ∀y∈domain(f). ∃z∈domain(f). ⟨x,z⟩∈r ∧ ⟨y,z⟩∈r
by blast
    ultimately have r directs domain(f) using IsDirectedSet_def by blast
  }
  moreover
  {
    have p:X∈𝔉 and 0∉𝔉 using assms IsFilter_def by auto
    then have X≠0 by auto
    then obtain q where q∈X by auto
    with p dom have ⟨q,X⟩∈domain(f) by auto
    then have domain(f)≠0 by blast
  }
  ultimately have ⟨f,r⟩ {is a net on}⋃𝔉 using IsNet_def by auto
  then show (Net(𝔉)) {is a net on} X using NetOfFilter_def assms uu by
auto
qed
```

If a filter converges to some point then its net converges to the same point.

```
theorem (in topology0) filter_conver_net_of_filter_conver:
  assumes 𝔉 {is a filter on} ⋃T and 𝔉 →F x
  shows (Net(𝔉)) →N x
proof-
  from assms have ⋃T∈𝔉 𝔉⊆Pow(⋃T) using IsFilter_def by auto
  then have uu: ⋃𝔉=⋃T by blast
  from assms(1) have func: fst(Net(𝔉))={⟨A,fst(A)⟩. A∈{⟨x,F⟩∈(⋃𝔉)×𝔉.
x∈F}}
    and dir: snd(Net(𝔉))={⟨A,B⟩∈{⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F}×{⟨x,F⟩∈(⋃𝔉)×𝔉.
x∈F}. snd(B)⊆snd(A)}
    using NetOfFilter_def uu by auto
  then have dom_def: domain(fst(Net(𝔉)))={⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F} by auto
  from func have fun: fst(Net(𝔉)): {⟨x,F⟩∈(⋃𝔉)×𝔉. x∈F} → (⋃𝔉)
    using ZF_fun_from_total by simp
  from assms(1) have NN: (Net(𝔉)) {is a net on}⋃T using net_of_filter_is_net
```

```
    by auto
  moreover from assms have x∈⋃T using FilterConverges_def
    by auto
  moreover
  {
    fix U
    assume AS: U∈Pow(⋃T) x∈int(U)
    with assms have U∈𝔉 x∈U using Top_2_L1 FilterConverges_def by auto
    then have pp: ⟨x,U⟩∈domain(fst(Net(𝔉))) using dom_def by auto
    {
      fix m
      assume ASS: m∈domain(fst(Net(𝔉))) ⟨⟨x,U⟩,m⟩∈snd(Net(𝔉))
      from ASS(1) fun func have fst(Net(𝔉))(m) = fst(m)
        using func1_1_L1 ZF_fun_from_tot_val by simp
      with dir ASS have fst(Net(𝔉))(m) ∈ U using dom_def by auto
    }
    then have ∀m∈domain(fst(Net(𝔉))). (⟨⟨x,U⟩,m⟩∈snd(Net(𝔉)) ⟶ fst(Net(𝔉))m∈U)
by auto
    with pp have ∃t∈domain(fst(Net(𝔉))). ∀m∈domain(fst(Net(𝔉))). (⟨t,m⟩∈snd(Net(𝔉))
⟶ fst(Net(𝔉))m∈U)
      by auto
  }
  then have ∀U∈Pow(⋃T).
      (x∈int(U) ⟶ (∃t∈domain(fst(Net(𝔉))). ∀m∈domain(fst(Net(𝔉))).
(⟨t,m⟩∈snd(Net(𝔉)) ⟶ fst(Net(𝔉))m∈U)))
      by auto
  ultimately show thesis using NetConverges_def by auto
qed
```

If a net converges to a point, then a filter also converges to a point.

```
theorem (in topology0) net_of_filter_conver_filter_conver:
  assumes 𝔉 {is a filter on}⋃T and (Net(𝔉)) →N x
  shows 𝔉 →F x
proof-
  from assms have ⋃T∈𝔉 𝔉⊆Pow(⋃T) using IsFilter_def by auto
  then have uu: ⋃𝔉=⋃T by blast
  have x∈⋃T using assms NetConverges_def net_of_filter_is_net by auto
  moreover
  {
    fix U
    assume U∈Pow(⋃T) x∈int(U)
    then obtain t where t: t∈domain(fst(Net(𝔉))) and
      reg: ∀m∈domain(fst(Net(𝔉))). ⟨t,m⟩∈snd(Net(𝔉)) ⟶ fst(Net(𝔉))m∈U
        using assms net_of_filter_is_net NetConverges_def by blast
    with assms(1) uu obtain t1 t2 where t_def: t=⟨t1,t2⟩ and t1∈t2 and
tFF: t2∈𝔉
      using NetOfFilter_def by auto
    {
      fix s
```

```
      assume s∈t2
      then have ⟨s,t2⟩∈{⟨q1,q2⟩∈⋃𝔉×𝔉. q1∈q2} using tFF by auto
      moreover
      from assms(1) uu have domain(fst(Net(𝔉)))={⟨q1,q2⟩∈⋃𝔉×𝔉. q1∈q2}
using NetOfFilter_def
        by auto
      ultimately
      have tt: ⟨s,t2⟩∈domain(fst(Net(𝔉))) by auto
      moreover
      from assms(1) uu t t_def tt have ⟨⟨t1,t2⟩,⟨s,t2⟩⟩∈snd(Net(𝔉)) us-
ing NetOfFilter_def
        by auto
      ultimately
      have fst(Net(𝔉))⟨s,t2⟩∈U using reg t_def by auto
      moreover
      from assms(1) uu have function(fst(Net(𝔉))) using NetOfFilter_def
function_def
        by auto
      moreover
      from tt assms(1) uu have ⟨⟨s,t2⟩,s⟩∈fst(Net(𝔉)) using NetOfFilter_def
by auto
      ultimately
      have s∈U using NetOfFilter_def function_apply_equality by auto
    }
    then have t2⊆U by auto
    with tFF assms(1) ‹U∈Pow(⋃T)› have U∈𝔉 using IsFilter_def by auto
  }
  then have {U∈Pow(⋃T). x∈int(U)} ⊆ 𝔉 by auto
  ultimately
  show thesis using FilterConverges_def assms(1) by auto
qed
```

A filter converges to a point if and only if its net converges to the point.

```
theorem (in topology0) filter_conver_iff_net_of_filter_conver:
  assumes 𝔉 {is a filter on}⋃T
  shows (𝔉 →F x) ⟷ ((Net(𝔉)) →N x)
  using filter_conver_net_of_filter_conver net_of_filter_conver_filter_conver
assms
    by auto
```

The previous result states that, when considering convergence, the filters do not generalize nets. When considering a filter, there is always a net that converges to the same points of the original filter.

Now we see that with nets, results come naturally applying the axiom of choice; but with filters, the results come, may be less natural, but with no choice. The reason is that Net(𝔉) is a net that doesn't come into our attention as a first choice; maybe because we restrict ourselves to the anti-symmetry property of orders without realizing that a directed set is not an order.

The following results will state that filters are not just a subclass of nets, but that nets and filters are equivalent on convergence: for every filter there is a net converging to the same points, and also, for every net there is a filter converging to the same points.

```
definition
  FilterOfNet (Filter (_ .. _) 40) where
  (N {is a net on} X) ⟹ Filter N..X ≡ {A∈Pow(X). ∃D∈{{fst(N)snd(s).
s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t0}}. t0∈domain(fst(N))}.
D⊆A}
```

Filter of a net is indeed a filter

```
theorem filter_of_net_is_filter:
  assumes N {is a net on} X
  shows (Filter N..X) {is a filter on} X and
    {{fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=t0}}.
t0∈domain(fst(N))} {is a base filter} (Filter N..X)
proof -
  let C = {{fst(N)(snd(s)). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t0}}. t0∈domain(fst(N))}
  have C⊆Pow(X)
  proof -
    {
      fix t
      assume t∈C
      then obtain t1 where t1∈domain(fst(N)) and
        t_Def: t={fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t1}}
        by auto
      {
        fix x
        assume x∈t
        with t_Def obtain ss where ss∈{s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=t1} and
          x_def: x = fst(N)(snd(ss)) by blast
        then have snd(ss) ∈ domain(fst(N)) by auto
        from assms have fst(N):domain(fst(N))→X unfolding IsNet_def
by simp
          with ‹snd(ss) ∈ domain(fst(N))› have x∈X using apply_funtype
x_def
        by auto
      }
      hence t⊆X by auto
    }
    thus thesis by blast
  qed
  have sat: C {satisfies the filter base condition}
  proof -
```

```
    from assms obtain t1 where t1∈domain(fst(N)) using IsNet_def by
blast
    hence {fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t1}}∈C
      by auto
    hence C≠0 by auto
    moreover
    {
      fix U
      assume U∈C
      then obtain q where q_dom: q∈domain(fst(N)) and
        U_def: U={fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=q}}
        by blast
      with assms have ⟨q,q⟩∈snd(N) ∧ fst(⟨q,q⟩)=q unfolding IsNet_def
IsDirectedSet_def refl_def
        by auto
      with q_dom have ⟨q,q⟩∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=q}
        by auto
      with U_def have fst(N)(snd(⟨q,q⟩)) ∈ U by blast
      hence U≠0 by auto
    }
    then have 0∉C by auto
    moreover
    have ∀A∈C. ∀B∈C. (∃D∈C. D⊆A∩B)
    proof
      fix A
      assume pA: A∈C
      show ∀B∈C. ∃D∈C. D⊆A∩B
      proof
        {
          fix B
          assume B∈C
          with pA obtain qA qB where per: qA∈domain(fst(N)) qB∈domain(fst(N))
and
            A_def: A={fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=qA}} and
            B_def: B={fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=qB}}
              by blast
          have dir: snd(N) directs domain(fst(N)) using assms IsNet_def
by auto
          with per obtain qD where ine: ⟨qA,qD⟩∈snd(N) ⟨qB,qD⟩∈snd(N)
and
          perD: qD∈domain(fst(N)) unfolding IsDirectedSet_def
            by blast
          let D = {fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=qD}}
```

```
          from perD have D∈C by auto
          moreover
          {
            fix d
            assume d∈D
            then obtain sd where sd∈{s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=qD} and
              d_def: d=fst(N)snd(sd) by blast
            then have sdN: sd∈snd(N) and qdd: fst(sd)=qD and sd∈domain(fst(N))×domain(fst

              by auto
            then obtain qI aa where sd = ⟨aa,qI⟩ qI ∈ domain(fst(N))
aa ∈ domain(fst(N))
              by auto
            with qdd have sd_def: sd=⟨qD,qI⟩ and qIdom: qI∈domain(fst(N))
by auto
            with sdN have ⟨qD,qI⟩∈snd(N) by auto
            from dir have trans(snd(N)) unfolding IsDirectedSet_def by
auto
            then have ⟨qA,qD⟩∈snd(N) ∧ ⟨qD,qI⟩∈snd(N) ⟶ ⟨qA,qI⟩∈snd(N)
and
              ⟨qB,qD⟩∈snd(N) ∧ ⟨qD,qI⟩∈snd(N)⟶⟨qB,qI⟩∈snd(N)
              using trans_def by auto
            with ine ‹⟨qD,qI⟩∈snd(N)› have ⟨qA,qI⟩∈snd(N) ⟨qB,qI⟩∈snd(N)
by auto
            with qIdom per have ⟨qA,qI⟩∈{s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=qA}
              ⟨qB,qI⟩∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=qB}

              by auto
            then have fst(N)(qI) ∈ A∩B using A_def B_def by auto
            then have fst(N)(snd(sd)) ∈ A∩B using sd_def by auto
            then have d ∈ A∩B using d_def by auto
          }
          then have D ⊆ A∩B by blast
          ultimately show ∃D∈C. D⊆A∩B by blast
        }
      qed
    qed
    ultimately
    show thesis unfolding SatisfiesFilterBase_def by blast
  qed
  have
    Base: C {is a base filter} {A∈Pow(X). ∃D∈C. D⊆A} ⋃{A∈Pow(X). ∃D∈C.
D⊆A}=X
  proof -
    from ‹C⊆Pow(X)› sat show C {is a base filter} {A∈Pow(X). ∃D∈C.
D⊆A}
      by (rule base_unique_filter_set3)
```

```
    from ‹C⊆Pow(X)› sat show ⋃{A∈Pow(X). ∃D∈C. D⊆A}=X
      by (rule base_unique_filter_set3)
  qed
  with sat show (Filter N..X) {is a filter on} X
    using sat basic_filter FilterOfNet_def assms by auto
  from Base(1) show C {is a base filter} (Filter N..X)
    using FilterOfNet_def assms by auto
qed
```

Convergence of a net implies the convergence of the corresponding filter.

```
theorem (in topology0) net_conver_filter_of_net_conver:
  assumes N {is a net on} ⋃T and N →N x
  shows (Filter N..(⋃T)) →F x
proof -
  let C = {{fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t}}.
      t∈domain(fst(N))}
  from assms(1) have
    (Filter N..(⋃T)) {is a filter on} (⋃T) and C {is a base filter}
Filter N..(⋃T)
    using filter_of_net_is_filter by auto
  moreover have ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)
  proof -
    {
      fix U
      assume U∈Pow(⋃T) x∈int(U)
      with assms have ∃t∈domain(fst(N)). (∀m∈domain(fst(N)). (⟨t,m⟩∈snd(N)
⟶ fst(N)m∈U))
        using NetConverges_def by auto
        then obtain t where t∈domain(fst(N)) and
          reg: ∀m∈domain(fst(N)). (⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U) by auto
      {
        fix f
        assume f∈{fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)).
s∈snd(N) ∧ fst(s)=t}}
        then obtain s where s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t} and
          f_def: f=fst(N)snd(s) by blast
        hence s∈domain(fst(N))×domain(fst(N)) and s∈snd(N) and fst(s)=t

          by auto
        hence s=⟨t,snd(s)⟩ and snd(s)∈domain(fst(N)) by auto
        with ‹s∈snd(N)› reg have fst(N)snd(s)∈U by auto
        with f_def have f∈U by auto
      }
      hence {fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t}} ⊆ U
        by blast
      with ‹t∈domain(fst(N))› have ∃D∈C. D⊆U
```

```
      by auto
    } thus ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)  by auto
  qed
  moreover from assms have x∈⋃T using NetConverges_def by auto
  ultimately show (Filter N..(⋃T)) →F x by (rule convergence_filter_base2)
qed
```

Convergence of a filter corresponding to a net implies convergence of the net.

```
 theorem (in topology0) filter_of_net_conver_net_conver:
  assumes N {is a net on} ⋃T and (Filter N..(⋃T)) →F x
  shows N →N x
proof -
  let C = {{fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=t}}.
      t∈domain(fst(N))}
  from assms have I: (Filter N..(⋃T)) {is a filter on} (⋃T)
    C {is a base filter} (Filter N..(⋃T)) (Filter N..(⋃T)) →F x
    using filter_of_net_is_filter by auto
  then have reg: ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈C. D⊆U)
    by (rule convergence_filter_base1)
  from I have x∈⋃T by (rule convergence_filter_base1)
  moreover
  {
    fix U
    assume U∈Pow(⋃T) x∈int(U)
    with reg have ∃D∈C. D⊆U by auto
    then obtain D where D∈C D⊆U
      by auto
    then obtain td where td∈domain(fst(N)) and
      D_def: D={fst(N)snd(s). s∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N)
∧ fst(s)=td}}
      by auto
    {
      fix m
      assume m∈domain(fst(N)) ⟨td,m⟩∈snd(N)
      with ‹td∈domain(fst(N))› have
        ⟨td,m⟩∈{s∈domain(fst(N))×domain(fst(N)). s∈snd(N) ∧ fst(s)=td}
        by auto
      with D_def have fst(N)m∈D by auto
      with ‹D⊆U› have fst(N)m∈U by auto
    }
    then have ∀m∈domain(fst(N)). ⟨td,m⟩∈snd(N) ⟶ fst(N)m∈U by auto
    with ‹td∈domain(fst(N))› have
      ∃t∈domain(fst(N)). ∀m∈domain(fst(N)). ⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U
      by auto
  }
  then have
    ∀U∈Pow(⋃T). x∈int(U) ⟶
```

```
      (∃t∈domain(fst(N)). ∀m∈domain(fst(N)). ⟨t,m⟩∈snd(N) ⟶ fst(N)m∈U)
      by auto
  ultimately show thesis using NetConverges_def assms(1) by auto
qed
```

Filter of net converges to a point x if and only the net converges to x.

```
theorem (in topology0) filter_of_net_conv_iff_net_conv:
  assumes N {is a net on} ⋃T
  shows ((Filter N..(⋃T)) →F x) ⟷ (N →N x)
  using assms filter_of_net_conver_net_conver net_conver_filter_of_net_conver

    by auto
```

We know now that filters and nets are the same thing, when working convergence of topological spaces. Sometimes, the nature of filters makes it easier to generalized them as follows.

Instead of considering all subsets of some set X, we can consider only open sets (we get an open filter) or closed sets (we get a closed filter). There are many more useful examples that characterize topological properties.

This type of generalization cannot be done with nets.

Also a filter can give us a topology in the following way:

```
theorem top_of_filter:
  assumes 𝔉 {is a filter on} ⋃𝔉
  shows (𝔉 ∪ {0}) {is a topology}
proof -
  {
    fix A B
    assume A∈(𝔉 ∪ {0})B∈(𝔉 ∪ {0})
    then have (A∈𝔉 ∧ B∈𝔉) ∨ (A∩B=0) by auto
    with assms have A∩B∈(𝔉 ∪ {0}) unfolding IsFilter_def
      by blast
  }
  then have ∀A∈(𝔉 ∪ {0}). ∀B∈(𝔉 ∪ {0}). A∩B∈(𝔉 ∪ {0}) by auto
  moreover
  {
    fix M
    assume A:M∈Pow(𝔉 ∪ {0})
    then have M=0∨M={0}∨(∃T∈M. T∈𝔉) by blast
    then have ⋃M=0∨(∃T∈M. T∈𝔉) by auto
    then obtain T where ⋃M=0∨(T∈𝔉 ∧ T∈M) by auto
    then have ⋃M=0∨(T∈𝔉 ∧ T⊆⋃M) by auto
    moreover from this A have ⋃M⊆⋃𝔉 by auto
    ultimately have ⋃M∈(𝔉∪{0}) using IsFilter_def assms by auto
    }
  then have ∀M∈Pow(𝔉∪{0}). ⋃M∈(𝔉∪{0}) by auto
  ultimately show thesis using IsATopology_def by auto
qed
```

We can use topology0 locale with filters.

```
lemma topology0_filter:
  assumes 𝔉 {is a filter on} ⋃𝔉
  shows topology0(𝔉 ∪ {0})
  using top_of_filter topology0_def assms by auto
```

The next abbreviation introduces notation where we want to specify the space where the filter convergence takes place.

```
abbreviation FilConvTop(_ →F _ {in} _)
  where 𝔉 →F x {in} T ≡ topology0.FilterConverges(T,𝔉,x)
```

The next abbreviation introduces notation where we want to specify the space where the net convergence takes place.

```
abbreviation NetConvTop(_ →N _ {in} _)
  where N →N x {in} T ≡ topology0.NetConverges(T,N,x)
```

Each point of a the union of a filter is a limit of that filter.

```
lemma lim_filter_top_of_filter:
  assumes 𝔉 {is a filter on} ⋃𝔉 and x∈⋃𝔉
  shows 𝔉 →F x {in} (𝔉∪{0})
proof-
  have ⋃𝔉=⋃(𝔉∪{0}) by auto
  with assms(1) have assms1: 𝔉 {is a filter on} ⋃(𝔉∪{0}) by auto
  {
    fix U
    assume U∈Pow(⋃(𝔉∪{0})) x∈Interior(U,(𝔉∪{0}))
    with assms(1) have Interior(U,(𝔉∪{0}))∈𝔉 using topology0_def top_of_filter
      topology0.Top_2_L2 by blast
    moreover
    from assms(1) have Interior(U,(𝔉∪{0}))⊆U using topology0_def top_of_filter
      topology0.Top_2_L1 by auto
    moreover
    from ‹U∈Pow(⋃(𝔉∪{0}))› have U∈Pow(⋃𝔉) by auto
    ultimately have U∈𝔉 using assms(1) IsFilter_def by auto
  }
  with assms assms1 show thesis using topology0.FilterConverges_def top_of_filter
    topology0_def by auto
qed

end
```

68 Topology and neighborhoods

```
theory Topology_ZF_4a imports Topology_ZF_4
begin
```

This theory considers the relations between topology and systems of neighborhood filters.

68.1 Neighborhood systems

The standard way of defining a topological space is by specifying a collection of sets that we consider "open" (see the `Topology_ZF` theory). An alternative of this approach is to define a collection of neighborhoods for each point of the space.

We define a neighborhood system as a function that takes each point $x \in X$ and assigns it a collection of subsets of X which is called the neighborhoods of x. The neighborhoods of a point x form a filter that satisfies an additional axiom that for every neighborhood N of x we can find another one U such that N is a neighborhood of every point of U.

```
definition
  IsNeighSystem (_ {is a neighborhood system on} _ 90)
  where 𝓜 {is a neighborhood system on} X ≡ (𝓜 : X→Pow(Pow(X))) ∧
  (∀x∈X. (𝓜(x) {is a filter on} X) ∧ (∀N∈𝓜(x). x∈N ∧ (∃U∈𝓜(x).∀y∈U.(N∈𝓜(y))
) ))
```

A neighborhood system on X consists of collections of subsets of X.

```
lemma neighborhood_subset:
  assumes 𝓜 {is a neighborhood system on} X and x∈X and N∈𝓜(x)
  shows N⊆X and x∈N
proof -
  from ‹𝓜 {is a neighborhood system on} X› have 𝓜 : X→Pow(Pow(X))
    unfolding IsNeighSystem_def by simp
  with ‹x∈X› have 𝓜(x) ∈ Pow(Pow(X)) using apply_funtype by blast
  with ‹N∈𝓜(x)› show N⊆X by blast
  from assms show x∈N using IsNeighSystem_def by simp
qed
```

Some sources (like Wikipedia) use a bit different definition of neighborhood systems where the U is required to be contained in N. The next lemma shows that this stronger version can be recovered from our definition.

```
lemma neigh_def_stronger:
  assumes 𝓜 {is a neighborhood system on} X and x∈X and N∈𝓜(x)
  shows ∃U∈𝓜(x).U⊆N ∧ (∀y∈U.(N∈𝓜(y)))
proof -
  from assms obtain W where W∈𝓜(x) and areNeigh:∀y∈W.(N∈𝓜(y))
    using  IsNeighSystem_def by blast
  let U = N∩W
  from assms ‹W∈𝓜(x)› have U ∈ 𝓜(x)
    unfolding IsNeighSystem_def IsFilter_def by blast
  moreover have U⊆N by blast
  moreover from areNeigh have ∀y∈U.(N∈𝓜(y)) by auto
  ultimately show thesis by auto
qed
```

68.2 From a neighborhood system to topology

Given a neighborhood system $\{\mathcal{M}_x\}_{x\in X}$ we can define a topology on X. Namely, we consider a subset of X open if $U \in \mathcal{M}_x$ for every element x of U.

The collection of sets defined as above is indeed a topology.

```
theorem topology_from_neighs:
  assumes 𝓜 {is a neighborhood system on} X
  defines Tdef: T ≡ {U∈Pow(X). ∀x∈U. U ∈ 𝓜(x)}
  shows T {is a topology} and ⋃T = X
proof -
  { fix 𝔘 assume 𝔘 ∈ Pow(T)
    have ⋃𝔘 ∈ T
    proof -
      from ‹𝔘 ∈ Pow(T)› Tdef have ⋃𝔘 ∈ Pow(X) by blast
      moreover
      { fix x assume x ∈ ⋃𝔘
        then obtain U where U∈𝔘 and x∈U by blast
        with assms ‹𝔘 ∈ Pow(T)›
        have U ∈ 𝓜(x) and U ⊆ ⋃𝔘 and  𝓜(x) {is a filter on} X
          unfolding IsNeighSystem_def by auto
        with ‹⋃𝔘 ∈ Pow(X)› have ⋃𝔘 ∈ 𝓜(x) unfolding IsFilter_def
          by simp
      }
      ultimately show ⋃𝔘 ∈ T using Tdef by blast
    qed
  }
  moreover
  { fix U V assume U∈T and V∈T
    have U∩V ∈ T
    proof -
      from Tdef ‹U∈T›  ‹U∈T› have U∩V ∈ Pow(X) by auto
      moreover
      { fix x assume x ∈ U∩V
        with assms ‹U∈T› ‹V∈T› Tdef have U ∈ 𝓜(x) V ∈ 𝓜(x) and  𝓜(x)
{is a filter on} X
          unfolding IsNeighSystem_def by auto
        then have U∩V ∈ 𝓜(x) unfolding IsFilter_def by simp
      }
      ultimately show U∩V ∈T using Tdef by simp
    qed
  }
  ultimately show T {is a topology} unfolding IsATopology_def by blast

  from assms show ⋃T = X unfolding IsNeighSystem_def IsFilter_def by
blast
qed
```

Some sources (like Wikipedia) define the open sets generated by a neighborhood system "as those sets containing a neighborhood of each of their points". The next lemma shows that this definition is equivalent to the one we are using.

```
lemma topology_from_neighs1:
  assumes ℳ {is a neighborhood system on} X
  shows {U∈Pow(X). ∀x∈U. U ∈ ℳ(x)} = {U∈Pow(X). ∀x∈U. ∃V ∈ ℳ(x).
V⊆U}
proof
  let T = {U∈Pow(X). ∀x∈U. U ∈ ℳ(x)}
  let S = {U∈Pow(X). ∀x∈U. ∃V ∈ ℳ(x). V⊆U}
  show S⊆T
  proof -
    { fix U assume U∈S
      then have U∈Pow(X) by simp
      moreover
      from assms ‹U∈S› ‹U∈Pow(X)› have ∀x∈U. U ∈ ℳ(x)
        unfolding IsNeighSystem_def IsFilter_def by blast
      ultimately have U∈T by auto
    } thus thesis by auto
  qed
  show T⊆S by auto
qed
```

68.3 From a topology to a neighborhood system

Once we have a topology T we can define a natural neighborhood system on $X = \bigcup T$. In this section we define such neighborhood system and prove its basic properties.

For a topology T we define a neighborhood system of T as a function that takes an $x \in X = \bigcup T$ and assigns it the collection of supersets of open sets containing x. We call that the "neighborhood system of T"

```
definition
  NeighSystem ({neighborhood system of} _ 91)
  where {neighborhood system of} T ≡ { ⟨x,{N∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆N)}⟩.
x ∈ ⋃T }
```

The way we defined the neighborhood system of T means that it is a function on $\bigcup T$.

```
lemma neigh_fun: shows ({neighborhood system of} T): ⋃T → Pow(Pow(⋃T))
proof -
  let X = ⋃T
  have ∀x∈X. {N∈Pow(X).∃U∈T.(x∈U ∧ U⊆N)} ∈ Pow(Pow(X))
    by blast
  then show thesis unfolding NeighSystem_def using ZF_fun_from_total
    by simp
```

qed

The value of the neighborhood system of T at $x \in \bigcup T$ is the collection of supersets of open sets containing x.

```
lemma neigh_val: assumes x∈⋃T
  shows ({neighborhood system of} T)(x) = {N∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆N)}
  using assms ZF_fun_from_tot_val1 unfolding NeighSystem_def
  by simp
```

The next lemma shows that open sets are members of (what we will prove later to be) the natural neighborhood system on $X = \bigcup T$.

```
lemma open_are_neighs:
  assumes U∈T x∈U
  shows x ∈ ⋃T and U ∈ {V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)}
  using assms by auto
```

Another fact we will need is that for every $x \in X = \bigcup T$ the neighborhoods of x form a filter

```
lemma neighs_is_filter:
  assumes T {is a topology} and x ∈ ⋃T
  defines Mdef: ℳ ≡ {neighborhood system of} T
  shows ℳ(x) {is a filter on} (⋃T)
proof -
  let X = ⋃T
  let 𝔉 = {V∈Pow(X).∃U∈T.(x∈U ∧ U⊆V)}
  have 0∉𝔉 by blast
  moreover have X∈𝔉
  proof -
    from assms ‹x∈X› have X ∈ Pow(X) X∈T and x∈X ∧ X⊆X using carr_open

      by auto
    hence ∃U∈T.(x∈U ∧ U⊆X) by auto
    thus thesis by auto
  qed
  moreover have ∀A∈𝔉. ∀B∈𝔉. A∩B ∈ 𝔉
  proof -
    { fix A B assume A∈𝔉 B∈𝔉
      then obtain U_A U_B where U_A∈T x∈U_A U_A⊆A U_B∈T x∈U_B U_B⊆B
        by auto
      with ‹T {is a topology}› ‹A∈𝔉› ‹B∈𝔉› have A∩B ∈ Pow(X) and
        U_A∩U_B ∈ T x ∈ U_A∩U_B U_A∩U_B ⊆ A∩B using IsATopology_def
        by auto
      hence A∩B ∈ 𝔉 by blast
    } thus thesis by blast
  qed
  moreover have ∀B∈𝔉. ∀C∈Pow(X). B⊆C ⟶ C∈𝔉
  proof -
    { fix B C assume B∈𝔉 C ∈ Pow(X) B⊆C
```

```
    then obtain U where U∈T and x∈U U⊆B by blast
    with ‹C ∈ Pow(X)› ‹B⊆C› have C∈𝔉 by blast
  } thus thesis by auto
 qed
 ultimately have 𝔉 {is a filter on} X unfolding IsFilter_def by blast
 with Mdef ‹x∈X› show ℳ(x) {is a filter on} X using ZF_fun_from_tot_val1
NeighSystem_def
   by simp
qed
```

The next theorem states that the the natural neighborhood system on $X = \bigcup T$ indeed is a neighborhood system.

```
theorem neigh_from_topology:
  assumes T {is a topology}
  shows ({neighborhood system of} T) {is a neighborhood system on} (⋃T)
proof -
  let X = ⋃T
  let ℳ = {neighborhood system of} T
  have ℳ : X→Pow(Pow(X))
  proof -
    { fix x assume x∈X
      hence {V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)} ∈ Pow(Pow(X)) by auto
    } hence ∀x∈X. {V∈Pow(⋃T).∃U∈T.(x∈U ∧ U⊆V)} ∈ Pow(Pow(X)) by auto
    then show thesis using ZF_fun_from_total NeighSystem_def by simp
  qed
  moreover from assms have ∀x∈X. (ℳ(x) {is a filter on} X)
    using neighs_is_filter NeighSystem_def by auto
  moreover have ∀x∈X. ∀N∈ℳ(x). x∈N ∧ (∃U∈ℳ(x).∀y∈U.(N∈ℳ(y)))
  proof -
    { fix x N assume x∈X N ∈ ℳ(x)
      let 𝔉 = {V∈Pow(X).∃U∈T.(x∈U ∧ U⊆V)}
      from ‹x∈X› have ℳ(x) = 𝔉 using ZF_fun_from_tot_val1 NeighSystem_def

        by simp
      with ‹N ∈ ℳ(x)› have N∈𝔉 by simp
      hence x∈N by blast
      from ‹N∈𝔉› obtain U where U∈T x∈U and U⊆N by blast
      with ‹N∈𝔉› ‹ℳ(x) = 𝔉› have U ∈ ℳ(x) by auto
      moreover from assms ‹U∈T›  ‹U⊆N› ‹N∈𝔉› have  ∀y∈U.(N ∈ ℳ(y))
        using ZF_fun_from_tot_val1 open_are_neighs neighs_is_filter
              NeighSystem_def IsFilter_def by auto
      ultimately have ∃U∈ℳ(x).∀y∈U.(N∈ℳ(y)) by blast
      with ‹x∈N› have x∈N ∧ (∃U∈ℳ(x).∀y∈U.(N∈ℳ(y))) by simp
    } thus thesis by auto
  qed
  ultimately show thesis unfolding IsNeighSystem_def by blast
qed
```

Any neighborhood of an element of the closure of a subset intersects the subset.

```
lemma neigh_inter_nempty:
  assumes T {is a topology} A⊆⋃T x ∈ Closure(A,T) and
  N ∈ ({neighborhood system of} T)(x)
  shows N∩A ≠ 0
proof -
  let X = ⋃T
  from assms(1) have cntx: topology0(T)
    unfolding topology0_def by simp
  with assms(2,3) have x∈X
    using topology0.Top_3_L11(1) by blast
  with assms(4) obtain U where U∈T x∈U and U⊆N
    using neigh_val by auto
  from assms(2,3) cntx ‹U∈T› ‹x∈U› have A∩U ≠ 0
    using topology0.cl_inter_neigh by simp
  with ‹U⊆N› show N∩A ≠ 0 by blast
qed
```

68.4 Neighborhood systems are 1:1 with topologies

We can create a topology from a neighborhood system and neighborhood system from topology. The question is then if we start from a neighborhood system, create a topology from it then create the topology's natural neighborhood system, do we get back the neighborhood system we started from? Similarly, if we start from a topology, create its neighborhood system and then create a topology from that, do we get the original topology? This section provides the affirmative answer (for now only for the first question). This means that there is a one-to-one correspondence between the set of topologies on a set and the set of abstract neighborhood systems on the set.

Each abstract neighborhood of x contains an open neighborhood of x.

```
lemma open_nei_in_nei:
  assumes 𝓜 {is a neighborhood system on} X x∈X N∈𝓜(x)
  defines Tdef: T ≡ {U∈Pow(X). ∀x∈U. U ∈ 𝓜(x)}
  shows N∈Pow(X) and ∃U∈T. (x∈U ∧ U⊆N)
proof -
  from assms(1) have 𝓜:X→Pow(Pow(X)) unfolding IsNeighSystem_def
    by simp
  with assms(2,3) show N∈Pow(X)using apply_funtype by blast
  let U = {y∈X. N ∈ 𝓜(y)}
  have U∈T
  proof -
    have U ∈ Pow(X) by auto
    moreover have ∀y∈U. U∈𝓜(y)
    proof -
      { fix y assume y∈U
        then have y∈X and N∈𝓜(y) by auto
        with assms(1) obtain V where V∈𝓜(y) and ∀z∈V. N ∈ 𝓜(z)
```

```
          unfolding IsNeighSystem_def by blast
        with assms(1) ‹y∈X› ‹V∈ℳ(y)› have V⊆U
          using neighborhood_subset(1) by blast
        with assms(1) ‹y∈X› ‹V∈ℳ(y)› ‹U ∈ Pow(X)› have U∈ℳ(y)
          unfolding IsNeighSystem_def using is_filter_def_split(5) by
blast
      } thus thesis by simp
    qed
    ultimately have U ∈ {U∈Pow(X). ∀x∈U. U ∈ ℳ(x)} by simp
    with assms(4) show U∈T by simp
  qed
  moreover from assms(1,2) ‹N∈ℳ(x)› have x∈U ∧ U ⊆ N
    using neighborhood_subset(2) by auto
  ultimately show ∃U∈T. (x∈U ∧ U⊆N) by (rule witness_exists)
qed
```

In the the next theorem we show that if we start from a neighborhood system, create a topology from it, then create it's natural neighborhood system, we get back the original neighborhood system.

```
theorem nei_top_nei_round_trip:
  assumes ℳ {is a neighborhood system on} X
  defines Tdef: T ≡ {U∈Pow(X). ∀x∈U. U ∈ ℳ(x)}
  shows ({neighborhood system of} T) = ℳ
proof -
  let M = {neighborhood system of} T
  from assms have T {is a topology} and ⋃T = X using topology_from_neighs

    by auto
  then have M {is a neighborhood system on} X using neigh_from_topology

    by blast
  with assms(1) have M:X→Pow(Pow(X)) and ℳ:X→Pow(Pow(X))
    unfolding IsNeighSystem_def by auto
  moreover
  { fix x assume x∈X
    from ‹⋃T = X› ‹x∈X› have I: M(x) = {V∈Pow(X).∃U∈T.(x∈U ∧ U⊆V)}
      unfolding NeighSystem_def using ZF_fun_from_tot_val1 by simp
    have M(x) = ℳ(x)
    proof
      { fix V assume V∈M(x)
        with I obtain U where U∈T x∈U U⊆V by auto
        from assms(2) ‹U∈T› ‹x∈U› have U ∈ ℳ(x) by simp
        from assms(1) ‹x∈X› have ℳ(x) {is a filter on} X
          unfolding IsNeighSystem_def by simp
        with ‹U ∈ ℳ(x)› ‹V∈M(x)› I ‹U⊆V›  have V ∈ ℳ(x)
          unfolding IsFilter_def by simp
      } thus M(x) ⊆ ℳ(x) by auto
      { fix N assume N∈ℳ(x)
        with assms ‹x∈X› ‹⋃T = X› I have N∈M(x) using open_nei_in_nei
```

```
        by auto
    } thus 𝓜(x) ⊆ M(x) by auto
  qed
} hence ∀x∈X. M(x) = 𝓜(x) by simp
ultimately show thesis by (rule func_eq)
qed
```

68.5 Set neighborhoods

Some sources (like Metamath) take a somewhat different approach where instead of defining the collection of neighborhoods of a point $x \in X$ they define a collection of neighborhoods of a subset of X (where X is the carrier of a topology T (i.e. $X = \bigcup T$). In this approach a neighborhood system is a function whose domain is the powerset of X, i.e. the set of subsets of X. The two approaches are equivalent in a sense as having a neighborhood system we can create a set neighborhood system and vice versa.

We define a set neighborhood system as a function that takes a subset A of the carrier of the topology and assigns it the collection of supersets of all open sets that contain A.

```
definition
  SetNeighSystem ( {set neighborhood system of} _ 91)
  where {set neighborhood system of} T
        ≡ {⟨A,{N∈Pow(⋃T). ∃U∈T. (A⊆U ∧ U⊆N)}⟩. A∈Pow(⋃T)}
```

Given a set neighborhood system we can recover the (standard) neighborhood system by taking the values of the set neighborhood system at singletons $\{x\}$ where $x \in X = \bigcup T$.

```
lemma neigh_from_nei: assumes x∈⋃T
  shows ({neighborhood system of} T)(x) = ({set neighborhood system of}
T){x}
  using assms ZF_fun_from_tot_val1
  unfolding NeighSystem_def SetNeighSystem_def
  by simp
```

The set neighborhood system of T is a function mapping subsets of $\bigcup T$ to collections of subsets of $\bigcup T$.

```
lemma nei_fun:
  shows ({set neighborhood system of} T):Pow(⋃T) →Pow(Pow(⋃T))
proof -
  let X = ⋃T
  have ∀A∈Pow(X). {N∈Pow(X). ∃U∈T. (A⊆U ∧ U⊆N)} ∈ Pow(Pow(X))
    by blast
  then have
    {⟨A,{N∈Pow(X). ∃U∈T. (A⊆U ∧ U⊆N)}⟩. A∈Pow(X)}:Pow(X)→Pow(Pow(X))
    by (rule ZF_fun_from_total)
```

```
  then show thesis unfolding SetNeighSystem_def by simp
qed
```

The value of the set neighborhood system of T at subset A of $\bigcup T$ is the collection of subsets N of $\bigcup T$ for which exists an open subset $U \subseteq N$ that contains A.

```
lemma nei_val: assumes A ⊆ ⋃T
  shows
  ({set neighborhood system of} T)(A) = {N∈Pow(⋃T). ∃U∈T. (A⊆U ∧ U⊆N)}
  using assms ZF_fun_from_tot_val1 unfolding SetNeighSystem_def by simp
```

A member of the value of the set neighborhood system of T at A is a subset of $\bigcup T$. The interesting part is that we can show it without any assumption on A.

```
lemma nei_val_subset:
  assumes N ∈ ({set neighborhood system of} T)(A)
  shows A ⊆ ⋃T and N ⊆ ⋃T
proof -
  let f = {set neighborhood system of} T
  have f:Pow(⋃T) →Pow(Pow(⋃T)) using nei_fun by simp
  with assms show A ⊆ ⋃T using arg_in_domain by blast
  with assms show N ⊆ ⋃T using nei_val by simp
qed
```

If T is a topology, then every subset of its carrier (i.e. $\bigcup T$) is a (set) neighborhood of the empty set.

```
lemma nei_empty: assumes T {is a topology} N ⊆ ⋃T
  shows N ∈ ({set neighborhood system of} T)(0)
  using assms empty_open nei_val by auto
```

If T is a topology, then the (set) neighborhoods of a nonempty subset of $\bigcup T$ form a filter on $X = \bigcup T$.

```
theorem nei_filter: assumes T {is a topology} D ⊆ (⋃T) D≠0
  shows ({set neighborhood system of} T)(D) {is a filter on} (⋃T)
proof -
  let X = ⋃T
  let 𝓕 = ({set neighborhood system of} T)(D)
  from assms(2) have I: 𝓕 = {N∈Pow(X). ∃U∈T. (D⊆U ∧ U⊆N)}
    using nei_val by simp
  with assms(3) have 0 ∉ 𝓕 by auto
  moreover from assms(1,2) I have X∈𝓕
    using carr_open by auto
  moreover from I have 𝓕 ⊆ Pow(X) by auto
  moreover have ∀A∈𝓕. ∀B∈𝓕. A∩B ∈ 𝓕
  proof -
    { fix A B assume A∈𝓕 B∈𝓕
      with I obtain U_A U_B where
        U_A∈T D⊆U_A U_A⊆A and U_B∈T D⊆U_B U_B⊆B
```

```
      by auto
    let U = U_A∩U_B
    from assms(1) ‹U_A∈T› ‹U_B∈T› ‹D⊆U_A› ‹D⊆U_B› ‹U_A⊆A› ‹U_B⊆B›
    have U ∈ T D⊆U U ⊆ A∩B
      unfolding IsATopology_def by auto
    with I ‹A∈𝓕› ‹B∈𝓕› have A∩B ∈ 𝓕 by auto
  } thus thesis by simp
qed
moreover have ∀B∈𝓕. ∀C∈Pow(X). B⊆C ⟶ C∈𝓕
proof -
  { fix B C assume B∈𝓕 C∈Pow(X) B⊆C
    from I ‹B∈𝓕› obtain U where U∈T D⊆U and U⊆B
      by auto
    with ‹B⊆C› have ∃U∈T. (D⊆U ∧ U⊆C) by blast
    with I ‹C∈Pow(X)› have C∈𝓕 by simp
  } thus thesis by blast
qed
ultimately show 𝓕 {is a filter on} X
  unfolding IsFilter_def by simp
qed
```

If N is a (set) neighborhood of A in T, then exist an open set U such that N contains U which contains A. This is similar to the Metamath's theorem with the same name, except that here we do not need assume that T is a topology (which is a bit worrying).

```
lemma neii2: assumes N ∈ ({set neighborhood system of} T)(A)
  shows ∃U∈T. (A⊆U ∧ U⊆N)
proof -
  from assms have A∈Pow(⋃T) using nei_fun arg_in_domain
    by blast
  with assms show thesis
    unfolding SetNeighSystem_def using ZF_fun_from_tot_val1
    by simp
qed
```

An open set U covering a set A is a set neighborhood of A.

```
lemma open_superset_nei: assumes V∈T A⊆V
  shows V ∈ ({set neighborhood system of} T)(A)
proof -
  from assms have
    ({set neighborhood system of} T)(A) = {N∈Pow(⋃T). ∃U∈T. (A⊆U ∧
U⊆N)}
    using nei_val by blast
  with assms show thesis by auto
qed
```

An open set is a set neighborhood of itself.

```
corollary open_is_nei: assumes V∈T
```

```
  shows V ∈ ({set neighborhood system of} T)(V)
  using assms open_superset_nei by simp
```

An open neighborhood of x is a set neighborhood of $\{x\}$.

```
corollary open_nei_singl: assumes V∈T x∈V
  shows  V ∈ ({set neighborhood system of} T){x}
  using assms open_superset_nei by simp
```

The Cartesian product of two neighborhoods is a neighborhood in the product topology. Similar to the Metamath's theorem with the same name.

```
lemma neitx:
  assumes T {is a topology} S {is a topology} and
  A ∈ ({set neighborhood system of} T)(C) and
  B ∈ ({set neighborhood system of} S)(D)
  shows A×B ∈ ({set neighborhood system of} (T×tS))(C×D)
proof -
  have A×B ⊆ ⋃(T×tS)
  proof -
    from assms(3,4) have A×B ⊆ (⋃T)×(⋃S)
      using nei_val_subset(2) by blast
    with assms(1,2) show thesis using Top_1_4_T1 by simp
  qed
  let ℱ = ({set neighborhood system of} (T×tS))(C×D)
  { assume C=0 ∨ D=0
    with assms(1,2) ‹A×B ⊆ ⋃(T×tS)› have A×B ∈ ℱ
      using Top_1_4_T1(1) nei_empty by auto
  }
  moreover
  { assume C≠0 D≠0
    from assms(3) obtain UA where
      UA∈T C⊆UA UA⊆A using neii2 by blast
    from assms(4) obtain UB where
      UB∈S D⊆UB UB⊆B using neii2 by blast
    from assms(1,2) ‹UA∈T› ‹UB∈S› ‹C⊆UA› ‹D⊆UB›
    have UA×UB ∈ T×tS and C×D ⊆ UA×UB
      using prod_open_open_prod by auto
    then have UA×UB ∈ ℱ using open_superset_nei
      by simp
    from ‹UA⊆A› ‹UB⊆B› have UA×UB ⊆ A×B by auto
    have ℱ {is a filter on} (⋃(T×tS))
    proof -
      from assms(1,2) have (T×tS) {is a topology}
        using Top_1_4_T1(1) by simp
      moreover have C×D ⊆ ⋃(T×tS)
      proof -
        from assms(3,4) have C×D ⊆ (⋃T)×(⋃S)
          using nei_val_subset(1) by blast
        with assms(1,2) show thesis using Top_1_4_T1(3) by simp
      qed
```

```
    moreover from ‹C≠0› ‹D≠0› have C×D ≠ 0 by auto
    ultimately show 𝓕 {is a filter on} (⋃(T×tS))
      using nei_filter by simp
  qed
  with ‹UA×UB ∈ 𝓕› ‹A×B ⊆ ⋃(T×tS)› ‹UA×UB ⊆ A×B›
  have A×B ∈ 𝓕 using is_filter_def_split(5) by simp
 }
 ultimately show thesis by auto
qed
```

Any neighborhood of an element of the closure of a subset intersects the subset. This is practically the same as `neigh_inter_nempty`, just formulated in terms of set neighborhoods of singletons. Compare with Metamath's theorem with the same name.

```
lemma neindisj: assumes T {is a topology} A⊆⋃T x ∈ Closure(A,T) and
  N ∈ ({set neighborhood system of} T){x}
  shows N∩A ≠ 0
proof -
  let X = ⋃T
  from assms(1) have cntx: topology0(T)
    unfolding topology0_def by simp
  with assms(2,3) have x∈X
    using topology0.Top_3_L11(1) by blast
  with assms show thesis using neigh_from_nei neigh_inter_nempty
    by simp
qed

end
```

69 Uniform spaces

```
theory UniformSpace_ZF imports Cardinal_ZF Order_ZF_1a Topology_ZF_2
Topology_ZF_4a
begin
```

This theory defines uniform spaces and proves their basic properties.

69.1 Entourages and neighborhoods

Just like a topological space constitutes the minimal setting in which one can speak of continuous functions, the notion of uniform spaces (commonly attributed to André Weil) captures the minimal setting in which one can speak of uniformly continuous functions. In some sense this is a generalization of the notion of metric (or metrizable) spaces and topological groups.

There are several definitions of uniform spaces. The fact that these definitions are equivalent is far from obvious (some people call such phenomenon cryptomorphism). We will use the definition of the uniform structure (or "uniformity") based on entourages. This was the original definition by Weil and it seems to be the most commonly used. A uniformity consists of entourages that are binary relations between points of space X that satisfy a certain collection of conditions, specified below.

```
definition
  IsUniformity (_ {is a uniformity on} _ 90) where
    Φ {is a uniformity on} X ≡(Φ {is a filter on} (X×X))
    ∧ (∀U∈Φ. id(X) ⊆ U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U) ∈ Φ)
```

Since the whole $X \times X$ is in a uniformity, a uniformity is never empty.

```
lemma uniformity_non_empty: assumes Φ {is a uniformity on} X
    shows Φ≠∅
  using assms unfolding IsUniformity_def IsFilter_def by auto
```

If Φ is a uniformity on X, then the every element V of Φ is a certain relation on X (a subset of $X \times X$) and is called an "entourage". For an $x \in X$ we call $V\{x\}$ a neighborhood of x. The first useful fact we will show is that neighborhoods are non-empty.

```
lemma neigh_not_empty:
  assumes Φ {is a uniformity on} X W∈Φ and x∈X
  shows W{x} ≠ ∅ and x ∈ W{x}
proof -
  from assms(1,2) have id(X) ⊆ W
    unfolding IsUniformity_def IsFilter_def by auto
  with ‹x∈X› show x∈W{x} and W{x} ≠ ∅ by auto
qed
```

The filter part of the definition of uniformity for easier reference:

```
lemma unif_filter: assumes Φ {is a uniformity on} X
  shows Φ {is a filter on} (X×X)
  using assms unfolding IsUniformity_def by simp
```

If X is not empty then the singleton $\{X \times X\}$ is a (trivial) example of a uniformity on X.

```
lemma min_uniformity: assumes X≠∅ shows {X×X} {is a uniformity on}
X
  using assms unfolding IsFilter_def IsUniformity_def by auto
```

On the other side of the spectrum is the collection of sets containing the diagonal, that is also a uniformity.

```
lemma max_uniformity: assumes X≠∅
  shows {U∈Pow(X×X). id(X)⊆U} {is a uniformity on} X
  using assms unfolding IsFilter_def IsUniformity_def by auto
```

The second part of the definition of uniformity for easy reference:

```
lemma entourage_props:
  assumes Φ {is a uniformity on} X and A∈Φ
  shows
    A ⊆ X×X
    id(X) ⊆ A
    ∃V∈Φ. V O V ⊆ A
    converse(A) ∈ Φ
proof -
  from assms show id(X) ⊆ A ∃V∈Φ. V O V ⊆ A converse(A) ∈ Φ
    unfolding IsUniformity_def by auto
  from assms show A ⊆ X×X
    using unif_filter unfolding IsFilter_def by blast
qed
```

The definition of uniformity states (among other things) that for every member U of uniformity Φ there is another one, say V such that $V \circ V \subseteq U$. Sometimes such V is said to be half the size of U. The next lemma states that V can be taken to be symmetric.

```
lemma half_size_symm: assumes Φ {is a uniformity on} X W∈Φ
  shows ∃V∈Φ. V O V ⊆ W ∧ V=converse(V)
proof -
  from assms obtain U where U∈Φ and U O U ⊆ W
    unfolding IsUniformity_def by auto
  let V = U ∩ converse(U)
  from assms(1) ‹U∈Φ› have V ∈ Φ and V = converse(V)
    unfolding IsUniformity_def IsFilter_def by auto
  moreover from ‹U O U ⊆ W› have V O V ⊆ W by auto
  ultimately show thesis by blast
qed
```

Inside every member W of the uniformity Φ we can find one that is symmetric and smaller than a third of size W. Compare with the Metamath's theorem with the same name.

```
lemma ustex3sym: assumes Φ {is a uniformity on} X A∈Φ
  shows ∃B∈Φ. B O (B O B) ⊆ A ∧ B=converse(B)
proof -
  from assms obtain C where C∈Φ and C O C ⊆ A
    unfolding IsUniformity_def by auto
  from assms(1) ‹C∈Φ› obtain B where
    B∈Φ B O B ⊆ C and B=converse(B)
    using half_size_symm by blast
  with ‹C O C ⊆ A› have (B O B) O (B O B) ⊆ A by blast
  with assms(1) ‹B∈Φ› have B O (B O B) ⊆ A
    using entourage_props(1,2) by blast
  with ‹B∈Φ› ‹B=converse(B)› show thesis by blast
qed
```

If Φ is a uniformity on X then every element of Φ is a subset of $X \times X$ whose domain is X.

```
lemma uni_domain:
  assumes Φ {is a uniformity on} X W∈Φ
  shows W ⊆ X×X and domain(W) = X
proof -
  from assms show W ⊆ X×X unfolding IsUniformity_def IsFilter_def
    by blast
  show domain(W) = X
  proof
    from assms show domain(W) ⊆ X unfolding IsUniformity_def IsFilter_def

      by auto
    from assms show X ⊆ domain(W) unfolding IsUniformity_def by blast
  qed
qed
```

If Φ is a uniformity on X and $W \in \Phi$ the for every $x \in X$ the image of the singleton $\{x\}$ by W is contained in X. Compare the Metamath's theorem with the same name.

```
lemma ustimasn:
  assumes Φ {is a uniformity on} X W∈Φ and x∈X
  shows W{x} ⊆ X
  using assms uni_domain(1) by auto
```

Uniformity Φ defines a natural topology on its space X via the neighborhood system that assigns the collection $\{V(\{x\}) : V \in \Phi\}$ to every point $x \in X$. In the next lemma we show that if we define a function this way the values of that function are what they should be. This is only a technical fact which is useful to shorten the remaining proofs, usually treated as obvious in standard mathematics.

```
lemma neigh_filt_fun:
  assumes Φ {is a uniformity on} X
  defines ℳ ≡ {⟨x,{V{x}.V∈Φ}⟩.x∈X}
  shows ℳ:X→Pow(Pow(X)) and ∀x∈X. ℳ(x) = {V{x}.V∈Φ}
proof -
  from assms have ∀x∈X. {V{x}.V∈Φ} ∈ Pow(Pow(X))
    using IsUniformity_def IsFilter_def image_subset by auto
  with assms show ℳ:X→Pow(Pow(X)) using ZF_fun_from_total by simp
  with assms show ∀x∈X. ℳ(x) = {V{x}.V∈Φ} using ZF_fun_from_tot_val
    by simp
qed
```

In the next lemma we show that the collection defined in lemma `neigh_filt_fun` is a filter on X. The proof is kind of long, but it just checks that all filter conditions hold.

```
lemma filter_from_uniformity:
  assumes Φ {is a uniformity on} X and x∈X
  defines ℳ ≡ {⟨x,{V{x}.V∈Φ}⟩.x∈X}
  shows ℳ(x) {is a filter on} X
```

```
proof -
  from assms have PhiFilter: Φ {is a filter on} (X×X) and
    ℳ:X→Pow(Pow(X)) and ℳ(x) = {V{x}.V∈Φ}
    using IsUniformity_def neigh_filt_fun by auto
  have ∅ ∉ ℳ(x)
  proof -
    from assms ‹x∈X› have ∅ ∉ {V{x}.V∈Φ} using neigh_not_empty by blast

    with ‹ℳ(x) = {V{x}.V∈Φ}› show ∅ ∉ ℳ(x) by simp
  qed
  moreover have X ∈ ℳ(x)
  proof -
    note ‹ℳ(x) = {V{x}.V∈Φ}›
    moreover from assms have X×X ∈ Φ unfolding IsUniformity_def IsFilter_def

      by blast
    hence (X×X){x} ∈ {V{x}.V∈Φ} by auto
    moreover from ‹x∈X› have (X×X){x} = X by auto
    ultimately show X ∈ ℳ(x) by simp
  qed
  moreover from ‹ℳ:X→Pow(Pow(X))› ‹x∈X› have ℳ(x) ⊆ Pow(X) using
apply_funtype
    by blast
  moreover have LargerIn: ∀B ∈ ℳ(x). ∀C ∈ Pow(X). B⊆C ⟶ C ∈ ℳ(x)
  proof -
  { fix B assume B ∈ ℳ(x)
    fix C assume C ∈ Pow(X) and B⊆C
    from ‹ℳ(x) = {V{x}.V∈Φ}› ‹B ∈ ℳ(x)› obtain U where
        U∈Φ and B = U{x} by auto
    let V = U ∪ C×C
    from assms ‹U∈Φ› ‹C ∈ Pow(X)› have V ∈ Pow(X×X) and U⊆V
      using IsUniformity_def IsFilter_def by auto
    with ‹U∈Φ› PhiFilter have V∈Φ using IsFilter_def by simp
    moreover from assms ‹U∈Φ› ‹x∈X› ‹B = U{x}› ‹B⊆C› have C = V{x}
      using neigh_not_empty image_greater_rel by simp
    ultimately have C ∈ {V{x}.V∈Φ} by auto
    with ‹ℳ(x) = {V{x}.V∈Φ}› have C ∈ ℳ(x) by simp
  } thus thesis by blast
  qed
  moreover have ∀A ∈ ℳ(x).∀B ∈ ℳ(x). A∩B ∈ ℳ(x)
  proof -
  { fix A B assume A ∈ ℳ(x) and B ∈ ℳ(x)
    with ‹ℳ(x) = {V{x}.V∈Φ}› obtain V_A V_B where
      A = V_A{x} B = V_B{x} and  V_A ∈ Φ V_B ∈ Φ
      by auto
    let C = V_A{x} ∩ V_B{x}
    from assms ‹V_A ∈ Φ› ‹V_B ∈ Φ› have V_A∩V_B ∈ Φ using IsUniformity_def
IsFilter_def
      by simp
```

```
    with ‹𝓜(x) = {V{x}.V∈Φ}› have (V_A∩V_B){x} ∈ 𝓜(x) by auto
    moreover from PhiFilter ‹V_A ∈ Φ› ‹V_B ∈ Φ› have C ∈ Pow(X) unfolding IsFilter_def
      by auto
    moreover have (V_A∩V_B){x} ⊆ C using image_Int_subset_left by simp
    moreover note LargerIn
    ultimately have C ∈ 𝓜(x) by simp
    with ‹A = V_A{x}› ‹B = V_B{x}› have A∩B ∈ 𝓜(x) by blast
  } thus thesis by simp
  qed
  ultimately show thesis unfolding IsFilter_def by simp
qed
```

A rephrasing of filter_from_uniformity: if Φ is a uniformity on X, then $\{V(\{x\}) | V \in \Phi\}$ is a filter on X for every $x \in X$.

```
lemma unif_filter_at_point:
  assumes Φ {is a uniformity on} X and x∈X
  shows {V{x}.V∈Φ} {is a filter on} X
  using assms filter_from_uniformity ZF_fun_from_tot_val1
  by simp
```

A frequently used property of filters is that they are "upward closed" i.e. supersets of a filter element are also in the filter. The next lemma makes this explicit for easy reference as applied to the natural filter created from a uniformity.

```
corollary unif_filter_up_closed:
  assumes Φ {is a uniformity on} X x∈X U ∈ {V{x}. V∈Φ} W⊆X U⊆W
  shows W ∈ {V{x}.V∈Φ}
  using assms filter_from_uniformity ZF_fun_from_tot_val1
  unfolding IsFilter_def by auto
```

The function defined in the premises of lemma neigh_filt_fun (or filter_from_uniformity) is a neighborhood system. The proof uses the existence of the "half-the-size" neighborhood condition (∃V∈Φ. V O V ⊆ U) of the uniformity definition, but not the converse(U) ∈ Φ part.

```
theorem neigh_from_uniformity:
  assumes Φ {is a uniformity on} X
  shows {⟨x,{V{x}.V∈Φ}⟩.x∈X} {is a neighborhood system on} X
proof -
  let 𝓜 = {⟨x,{V{x}.V∈Φ}⟩.x∈X}
  from assms have 𝓜:X→Pow(Pow(X)) and Mval: ∀x∈X. 𝓜(x) = {V{x}.V∈Φ}
    using IsUniformity_def neigh_filt_fun by auto
  moreover from assms have ∀x∈X. (𝓜(x) {is a filter on} X) using filter_from_uniformity
    by simp
  moreover
  { fix x assume x∈X
    have ∀N∈𝓜(x). x∈N ∧ (∃U∈𝓜(x).∀y∈U.(N∈𝓜(y)))
    proof -
```

```
    { fix N assume N∈ℳ(x)
      have x∈N and ∃U∈ℳ(x).∀y∈U.(N ∈ ℳ(y))
      proof -
        from ‹ℳ:X→Pow(Pow(X))› Mval ‹x∈X› ‹N∈ℳ(x)›
        obtain U where U∈Φ and N = U{x} by auto
        with assms ‹x∈X› show x∈N using neigh_not_empty by simp
        from assms ‹U∈Φ› obtain V where V∈Φ  and  V O V ⊆ U
          unfolding IsUniformity_def by auto
        let W = V{x}
        from ‹V∈Φ› Mval ‹x∈X› have W ∈ ℳ(x) by auto
        moreover have ∀y∈W. N ∈ ℳ(y)
        proof -
          { fix y assume y∈W
            with ‹ℳ:X→Pow(Pow(X))› ‹x∈X› ‹W ∈ ℳ(x)› have y∈X
              using apply_funtype by blast
            with assms have ℳ(y) {is a filter on} X using filter_from_uniformity
              by simp
            moreover from assms ‹y∈X› ‹V∈Φ› have V{y} ∈ ℳ(y)
              using neigh_filt_fun by auto
            moreover from ‹ℳ:X→Pow(Pow(X))› ‹x∈X› ‹N ∈ ℳ(x)›
              have N ∈ Pow(X) using apply_funtype by blast
            moreover from ‹V O V ⊆ U› ‹y∈W› have
              V{y} ⊆ (V O V){x} and (V O V){x} ⊆ U{x}
              by auto
            with ‹N = U{x}›  have V{y} ⊆ N by blast
            ultimately have N ∈ ℳ(y) unfolding IsFilter_def by simp
          } thus thesis by simp
        qed
        ultimately show ∃U∈ℳ(x).∀y∈U.(N ∈ ℳ(y)) by auto
      qed
    } thus thesis by simp
  qed
}
ultimately show thesis unfolding IsNeighSystem_def by simp
qed
```

When we have a uniformity Φ on X we can define a topology on X in a (relatively) natural way. We will call that topology the `UniformTopology(Φ)`. We could probably reformulate the definition to skip the X parameter because if Φ is a uniformity on X then X can be recovered from (is determined by) Φ.

```
definition
  UniformTopology(Φ,X) ≡ {U∈Pow(X). ∀x∈U. U∈{V{x}. V∈Φ}}
```

An identity showing how the definition of uniform topology is constructed. Here, the $M = \{\langle t, \{V\{t\} : V \in \Phi\}\rangle : t \in X\}$ is the neighborhood system (a function on X) created from uniformity Φ. Then for each $x \in X$, $M(x) = \{V\{t\} : V \in \Phi\}$ is the set of neighborhoods of x.

```
lemma uniftop_def_alt:
  shows UniformTopology(Φ,X) = {U ∈ Pow(X). ∀x∈U. U ∈ {⟨t,{V{t}.V∈Φ}⟩.t∈X}(x)}
proof -
  let ℳ = {⟨x,{V{x}. V∈Φ}⟩. x∈X}
  have ∀U∈Pow(X).∀x∈U. ℳ(x) = {V{x}. V∈Φ}
    using ZF_fun_from_tot_val1 by auto
  then show thesis unfolding UniformTopology_def by auto
qed
```

The collection of sets constructed in the `UniformTopology` definition is indeed a topology on X.

```
theorem uniform_top_is_top:
  assumes Φ {is a uniformity on} X
  shows
  UniformTopology(Φ,X) {is a topology} and ⋃ UniformTopology(Φ,X) =
X
  using assms neigh_from_uniformity uniftop_def_alt topology_from_neighs
  by auto
```

If we have a uniformity Φ we can create a neighborhood system from it in two ways. We can create a a neighborhood system directly from Φ using the formula $X \ni x \mapsto \{V\{x\} | V \in \Phi\}$ (see theorem `neigh_from_uniformity`). Alternatively we can construct a topology from Φ as in theorem `uniform_top_is_top` and then create a neighborhood system from this topology as in theorem `neigh_from_topology`. The next theorem states that these two ways give the same result.

```
theorem neigh_unif_same: assumes Φ {is a uniformity on} X
  shows
    {⟨x,{V{x}.V∈Φ}⟩. x∈X} = {neighborhood system of} UniformTopology(Φ,X)
  using assms neigh_from_uniformity nei_top_nei_round_trip uniftop_def_alt
  by simp
```

Another form of the definition of topology generated from a uniformity.

```
lemma uniftop_def_alt1: assumes Φ {is a uniformity on} X
  shows UniformTopology(Φ,X) = {U∈Pow(X).  ∀x∈U. ∃W∈Φ. W{x} ⊆ U}
proof
  let T = UniformTopology(Φ,X)
  show T ⊆ {U∈Pow(X).  ∀x∈U. ∃W∈Φ. W{x} ⊆ U}
    unfolding UniformTopology_def by auto
  { fix U assume U∈{U∈Pow(X).  ∀x∈U. ∃W∈Φ. W{x} ⊆ U}
    then have U∈Pow(X) and I: ∀x∈U. ∃W∈Φ. W{x} ⊆ U by auto
    { fix x assume x∈U
      with I obtain W where W∈Φ and W{x} ⊆ U
        by auto
      let 𝔉 = {V{x}.V∈Φ}
      from assms(1) ‹U∈Pow(X)› ‹x∈U› ‹W∈Φ› have
        𝔉 {is a filter on} X and W{x} ∈ 𝔉
        using unif_filter_at_point by auto
```

```
      with ‹U∈Pow(X)› ‹W{x} ⊆ U› have U∈𝔉
        using is_filter_def_split(5) by blast
    } hence ∀x∈U. U ∈ {V{x}.V∈Φ} by simp
    with ‹U∈Pow(X)› have U∈T
      unfolding UniformTopology_def by simp
  } thus {U∈Pow(X). ∀x∈U. ∃W∈Φ. W{x} ⊆ U} ⊆ T
    by blast
qed
```

Images of singletons by entourages are neighborhoods of those singletons.

```
lemma image_singleton_ent_nei:
  assumes Φ {is a uniformity on} X V∈Φ x∈X
  defines ℳ ≡ {neighborhood system of} UniformTopology(Φ,X)
  shows V{x} ∈ ℳ(x)
proof -
  from assms(1,4) have ℳ = {⟨x,{V{x}.V∈Φ}⟩. x∈X}
    using neigh_unif_same by simp
  with assms(2,3) show thesis
    using ZF_fun_from_tot_val1 by auto
qed
```

The set neighborhoods of a singleton $\{x\}$ where $x \in X$ consist of images of the singleton by the entourages $W \in \Phi$. See also the Metamath's theorem with the same name.

```
lemma utopsnneip: assumes Φ {is a uniformity on} X x∈X
  defines 𝒮 ≡ {set neighborhood system of} UniformTopology(Φ,X)
  shows 𝒮{x} = {W{x}. W∈Φ}
proof -
  let T = UniformTopology(Φ,X)
  let ℳ = {neighborhood system of} T
  from assms(1,2) have x ∈ ⋃T
    using uniform_top_is_top(2) by simp
  with assms(3) have ℳ(x) = 𝒮{x}
    using neigh_from_nei by simp
  moreover
  from assms(1) have ℳ = {⟨x,{W{x}.W∈Φ}⟩. x∈X}
    using neigh_unif_same by simp
  with assms(2) have ℳ(x) =  {W{x}.W∈Φ}
    using ZF_fun_from_tot_val1 by simp
  ultimately show thesis by simp
qed
```

Images of singletons by entourages are set neighborhoods of those singletons. See also the Metamath theorem with the same name.

```
corollary utopsnnei: assumes Φ {is a uniformity on} X W∈Φ x∈X
  defines 𝒮 ≡ {set neighborhood system of} UniformTopology(Φ,X)
  shows W{x} ∈ 𝒮{x} using assms utopsnneip by auto
```

If Φ is a uniformity on X that generates a topology T, R is any relation on

X (i.e. $R \subseteq X \times X$), W is a symmetric entourage (i.e. $W \in \Phi$, and W is symmetric (i.e. equal to its converse)), then the closure of R in the product topology is contained the the composition $V \circ (M \circ V)$. Metamath has a similar theorem with the same name.

```
lemma utop3cls:
  assumes Φ {is a uniformity on} X R⊆X×X W∈Φ W=converse(W)
  defines J ≡ UniformTopology(Φ,X)
  shows Closure(R,J×tJ) ⊆ W O (R O W)
proof
  let M = {set neighborhood system of} (J×tJ)
  fix z assume zMem: z ∈ Closure(R,J×tJ)
  from assms(1,5) have Jtop: J {is a topology} and ⋃J = X
    using uniform_top_is_top by auto
  then have JJtop: (J×tJ) {is a topology} and JxJ: ⋃(J×tJ) = X×X
    using Top_1_4_T1(1,3) by auto
  with assms(2) have topology0(J×tJ) and R ⊆ ⋃(J×tJ)
    unfolding topology0_def by auto
  then have Closure(R,J×tJ) ⊆ ⋃(J×tJ)
    using topology0.Top_3_L11(1) by simp
  with ‹z ∈ Closure(R,J×tJ)› JxJ have z∈X×X by auto
  let x = fst(z)
  let y = snd(z)
  from ‹z∈X×X› have x∈X y∈X z = ⟨x,y⟩ by auto
  with assms(1,3,5) Jtop have (W{x})×(W{y}) ∈ M({x}×{y})
    using utopsnnei neitx by simp
  moreover from ‹z = ⟨x,y⟩› have {x}×{y} = {z}
    by (rule pair_prod)
  ultimately have (W{x})×(W{y}) ∈ M{z} by simp
  with zMem JJtop ‹R ⊆ ⋃(J×tJ)› have (W{x})×(W{y}) ∩ R ≠ ∅
    using neindisj by blast
  with assms(4) have ⟨x,y⟩ ∈ W O (R O W)
    using sym_rel_comp by simp
  with ‹z = ⟨x,y⟩› show z ∈ W O (R O W)
    by simp
qed
```

Uniform spaces are regular. Note that is not the same as T_3, see `Topology_ZF_1` for separation axioms definitions. In some sources the definitions of "regular" and T_3 are swapped. In IsarMathLib we adopt the terminology as on the "Separation axiom" page on Wikipedia.

```
theorem utopreg:
  assumes Φ {is a uniformity on} X
  shows UniformTopology(Φ,X) {is regular}
proof -
  let J = UniformTopology(Φ,X)
  let 𝒮 = {set neighborhood system of} J
  from assms have ⋃J = X
    and Jtop: J {is a topology} and cntx: topology0(J)
```

```
    using uniform_top_is_top unfolding topology0_def by auto
  have ∀U∈J. ∀x∈U. ∃V∈J. x∈V ∧ Closure(V,J)⊆U
  proof -
    { fix U x assume U∈J x∈U
      then have U ∈ 𝒮{x} using open_nei_singl by simp
      from ‹U∈J› have U⊆⋃J by auto
      with ‹x∈U› ‹⋃J = X› have x∈X by auto
      from assms(1) ‹x∈X› ‹U ∈ 𝒮{x}› obtain A
        where U=A{x} and A∈Φ
        using utopsnneip by auto
      from assms(1) ‹A∈Φ› obtain W where
        W∈Φ W O (W O W) ⊆ A and Wsymm: W=converse(W)
        using ustex3sym by blast
      with assms(1) ‹x∈X› have W{x} ∈ 𝒮{x} and W{x} ⊆ X
        using utopsnnei ustimasn by auto
      from ‹W{x} ∈ 𝒮{x}› have ∃V∈J. {x}⊆V ∧ V⊆W{x}
        by (rule neii2)
      then obtain V where V∈J x∈V V⊆W{x}
        by blast
      have Closure(V,J) ⊆ U
      proof -
        from assms(1) ‹W∈Φ› ‹⋃J = X› have W ⊆ X×X
          using entourage_props(1) by simp
        from cntx ‹W{x} ⊆ X› ‹⋃J = X› ‹V⊆W{x}›
          have Closure(V,J) ⊆ Closure(W{x},J)
            using topology0.top_closure_mono by simp
        also have Closure(W{x},J) ⊆ Closure(W,J×ₜJ){x}
        proof -
          from ‹W⊆X×X› ‹x∈X› ‹⋃J = X›
            have W⊆(⋃J)×(⋃J) x∈⋃J by auto
          with ‹J {is a topology}› show thesis
            using imasncls by simp
        qed
        also from assms(1) ‹W⊆X×X› ‹W∈Φ› Wsymm ‹W O (W O W) ⊆ A›
          have Closure(W,J×ₜJ){x} ⊆ A{x}
            using utop3cls by blast
        finally have Closure(V,J) ⊆ A{x}
          by simp
        with ‹U=A{x}› show thesis by auto
      qed
      with ‹V∈J› ‹x∈V› have ∃V∈J. x∈V ∧ Closure(V,J)⊆U
        by blast
    } thus thesis by simp
  qed
  with Jtop show J {is regular} using is_regular_def_alt
    by simp
qed
```

If the topological space generated by a uniformity Φ on X is T_1 then the intersection $\bigcup\Phi$ is contained in the diagonal $\{\langle x,x\rangle : x \in X\}$. Note the opposite inclusion is true for every uniformity.

```
lemma unif_t1_inter_diag:
  assumes Φ {is a uniformity on} X and UniformTopology(Φ,X) {is T₁}
  shows ⋂Φ ⊆ {⟨x,x⟩. x∈X}
proof -
  let T = UniformTopology(Φ,X)
  from assms(1) have ⋂Φ ⊆ X×X using uniformity_non_empty entourage_props(1)

    by blast
  { fix p assume p ∈ ⋂Φ and p ∉ {⟨x,x⟩. x∈X}
    from ‹⋂Φ ⊆ X×X› ‹p ∈ ⋂Φ› obtain x y where x∈X y∈X p = ⟨x,y⟩ by
auto
    with assms ‹p ∉ {⟨x,x⟩. x∈X}› obtain U where U ∈ T x∈U y∉U
      using uniform_top_is_top(2) unfolding isT1_def by force
    with assms(1) ‹p = ⟨x,y⟩› ‹p ∈ ⋂Φ› have False
      using uniftop_def_alt1 by force
  } with ‹⋂Φ ⊆ X×X› show thesis by blast
qed
```

If the intersection of a uniformity is contained in the the diagonal $\{\langle x,x\rangle : x \in X\}$ then the topological space generated by this uniformity is T_1.

```
lemma unif_inter_diag_t1:
  assumes Φ {is a uniformity on} X and ⋂Φ ⊆ {⟨x,x⟩. x∈X}
  shows UniformTopology(Φ,X) {is T₁}
proof -
  let T = UniformTopology(Φ,X)
  let 𝓜 = {neighborhood system of} T
  from assms(1) have ⋃T = X using uniform_top_is_top(2) by simp
  { fix x y assume x∈X y∈X x≠y
    with assms obtain W where W∈Φ and y∉W{x}
      using uniformity_non_empty by blast
    from assms(1) ‹x∈X› ‹W∈Φ› have W{x} ∈ 𝓜(x)
      using image_singleton_ent_nei by simp
    with ‹x∈X› ‹⋃T = X› ‹y∉W{x}› have ∃U∈T. x∈U ∧ y∉U
      using neigh_val by auto
  } with ‹⋃T = X› show thesis unfolding isT1_def by simp
qed
```

If Φ is a uniformity on X then the intersection of Φ is contained in diagonal of X if and only if $\bigcup\Phi$ is equal to that diagonal. Some people call such uniform space "separating".

```
theorem unif_inter_diag: assumes Φ {is a uniformity on} X
  shows ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ ⋂Φ = {⟨x,x⟩. x∈X}
  using assms entourage_props(2) uniformity_non_empty by force
```

The next theorem collects the information we have to show that if Φ is a uniformity on X, with the induced topology T then conditions T is T_0, T is

T_1, T is T_2 T is T_3 are all equivalent to the intersection of Φ being contained in the diagonal (which is equivalent to the intersection of Φ being equal to the diagonal, see `unif_inter_diag` above.

```
theorem unif_sep_axioms_diag: assumes Φ {is a uniformity on} X
  defines T ≡ UniformTopology(Φ,X)
  shows
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₀}
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₁}
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₂}
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₃}
proof -
  from assms show ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₁}
    using unif_t1_inter_diag unif_inter_diag_t1 by auto
  with assms show
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₀}
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₂}
    ⋂Φ ⊆ {⟨x,x⟩. x∈X} ⟷ T {is T₃}
    using utopreg T1_is_T0 T3_is_T2 T2_is_T1
    unfolding isT3_def by auto
qed
```

69.2 Base of a uniformity

A `base` or a `fundamental system of entourages` of a uniformity Φ is a subset of Φ that is sufficient to uniquely determine it. This is analogous to the notion of a base of a topology (see `Topology_ZF_1` or a base of a filter (see `Topology_ZF_4`).

A base of a uniformity Φ is any subset $\mathfrak{B} \subseteq \Phi$ such that every entourage in Φ contains (at least) one from $\mathfrak{B}$. The phrase `is a base for` is already defined to mean a base for a topology, so we use the phrase `is a uniform base of` here.

```
definition
  IsUniformityBase (_ {is a uniform base of} _ 90) where
    𝔅 {is a uniform base of} Φ ≡ 𝔅 ⊆ Φ ∧ (∀U∈Φ. ∃B∈𝔅. B⊆U)
```

Symmetric entourages form a base of the uniformity.

```
lemma symm_are_base: assumes Φ {is a uniformity on} X
  shows {V∈Φ. V = converse(V)} {is a uniform base of} Φ
proof -
  let 𝔅 = {V∈Φ. V = converse(V)}
  { fix W assume W∈Φ
    with assms obtain V where V∈Φ V O V ⊆ W  V=converse(V)
      using half_size_symm by blast
    from assms ‹V∈Φ› have V ⊆ V O V
      using entourage_props(1,2) refl_square_greater by blast
    with ‹V O V ⊆ W› ‹V∈Φ› ‹V=converse(V)› have ∃V∈𝔅. V⊆W by auto
```

```
  } hence ∀W∈Φ. ∃V∈𝔅. V ⊆ W by auto
  then show thesis unfolding IsUniformityBase_def by auto
qed
```

Given a base of a uniformity we can recover the uniformity taking the supersets. The `Supersets` constructor is defined in `ZF1`.

```
lemma uniformity_from_base:
  assumes Φ {is a uniformity on} X 𝔅 {is a uniform base of} Φ
  shows Φ = Supersets(X×X,𝔅)
proof
  from assms show Φ ⊆ Supersets(X×X,𝔅)
    unfolding IsUniformityBase_def Supersets_def
    using entourage_props(1) by auto
  from assms show Supersets(X×X,𝔅) ⊆ Φ
    unfolding Supersets_def IsUniformityBase_def IsUniformity_def IsFilter_def
    by auto
qed
```

Analogous to the predicate "satisfies base condition" (defined in `Topology_ZF_1`) and "is a base filter" (defined in `Topology_ZF_4`) we can specify conditions for a collection $\mathfrak{B}$ of subsets of $X \times X$ to be a base of some uniformity on X. Namely, the following conditions are necessary and sufficient:

1. Intersection of two sets of $\mathfrak{B}$ contains a set of $\mathfrak{B}$.
2. Every set of $\mathfrak{B}$ contains the diagonal of $X \times X$.
3. For each set $B_1 \in \mathfrak{B}$ we can find a set $B_2 \in \mathfrak{B}$ such that $B_2 \subseteq B_1^{-1}$.
4. For each set $B_1 \in \mathfrak{B}$ we can find a set $B_2 \in \mathfrak{B}$ such that $B_2 \circ B_2 \subseteq B_1$.

The conditions in the definition below are taken from N. Bourbaki "Elements of Mathematics, General Topology", Chapter II.1., except for the last two that are missing there.

```
definition
  IsUniformityBaseOn (_ {is a uniform base on} _ 90) where
    𝔅 {is a uniform base on} X ≡
      (∀B₁∈𝔅. ∀B₂∈𝔅. ∃B₃∈𝔅. B₃⊆B₁∩B₂) ∧ (∀B∈𝔅. id(X)⊆B) ∧
      (∀B₁∈𝔅. ∃B₂∈𝔅. B₂ ⊆ converse(B₁)) ∧ (∀B₁∈𝔅. ∃B₂∈𝔅. B₂ O B₂
⊆ B₁) ∧
      𝔅⊆Pow(X×X) ∧ 𝔅≠∅
```

The next lemma splits the definition of `IsUniformityBaseOn` into four conditions to enable more precise references in proofs.

```
lemma uniformity_base_props: assumes 𝔅 {is a uniform base on} X
  shows
  ∀B₁∈𝔅. ∀B₂∈𝔅. ∃B₃∈𝔅. B₃⊆B₁∩B₂
  ∀B∈𝔅. id(X)⊆B
  ∀B₁∈𝔅. ∃B₂∈𝔅. B₂ ⊆ converse(B₁)
  ∀B₁∈𝔅. ∃B₂∈𝔅. B₂ O B₂ ⊆ B₁
  𝔅⊆Pow(X×X) and 𝔅≠∅
```

```
  using assms unfolding IsUniformityBaseOn_def by simp_all
```

If supersets of some collection of subsets of $X \times X$ form a uniformity, then this collection satisfies the conditions in the definition of IsUniformityBaseOn.

```
theorem base_is_uniform_base:
  assumes 𝔅⊆Pow(X×X) and Supersets(X×X,𝔅) {is a uniformity on} X
  shows 𝔅 {is a uniform base on} X
proof -
  let Φ = Supersets(X×X,𝔅)
  have ∀B₁∈𝔅. ∀B₂∈𝔅. ∃B₃∈𝔅. B₃⊆B₁∩B₂
  proof -
    { fix B₁ B₂ assume B₁∈𝔅 B₂∈𝔅
      with assms(1) have B₁∈Φ and B₂∈Φ unfolding Supersets_def by auto
      with assms(2) have ∃B₃∈𝔅. B₃ ⊆ B₁∩B₂
        unfolding IsUniformity_def IsFilter_def Supersets_def by simp
    } thus thesis by simp
  qed
  moreover have ∀B∈𝔅. id(X)⊆B
  proof -
    { fix B assume B∈𝔅
      with assms(1) have B∈Φ unfolding Supersets_def by auto
      with assms(2) have id(X)⊆B unfolding IsUniformity_def by simp
    } thus thesis by simp
  qed
  moreover have ∀B₁∈𝔅. ∃B₂∈𝔅. B₂ ⊆ converse(B₁)
  proof -
    { fix B₁ assume B₁∈𝔅
      with assms(1) have B₁∈Φ unfolding Supersets_def by auto
      with assms have ∃B₂∈𝔅. B₂ ⊆ converse(B₁)
        unfolding IsUniformity_def Supersets_def by auto
    } thus thesis by simp
  qed
  moreover have ∀B₁∈𝔅. ∃B₂∈𝔅. B₂ O B₂ ⊆ B₁
  proof -
    { fix B₁ assume B₁∈𝔅
      with assms(1) have B₁∈Φ unfolding Supersets_def by auto
      with assms(2) obtain V where V∈Φ and V O V ⊆ B₁
        unfolding IsUniformity_def by blast
      from assms(2) ‹V∈Φ› obtain B₂ where B₂∈𝔅 and B₂⊆V
        unfolding Supersets_def by auto
      from ‹V O V ⊆ B₁› ‹B₂⊆V› have B₂ O B₂ ⊆ B₁ by auto
      with ‹B₂∈𝔅› have ∃B₂∈𝔅. B₂ O B₂ ⊆ B₁ by auto
    } thus thesis by simp
  qed
  moreover from assms(2) have 𝔅≠∅
    using supersets_of_empty uniformity_non_empty by blast
  ultimately show 𝔅 {is a uniform base on} X
    unfolding IsUniformityBaseOn_def using assms(1) by simp
qed
```

if a nonempty collection of subsets of $X \times X$ satisfies conditions in the definition of `IsUniformityBaseOn` then the supersets of that collection form a uniformity on X.

```
theorem uniformity_base_is_base:
  assumes X≠∅ and 𝔅 {is a uniform base on} X
  shows Supersets(X×X,𝔅) {is a uniformity on} X
proof -
  let Φ = Supersets(X×X,𝔅)
  from assms(2) have 𝔅⊆Pow(X×X) using uniformity_base_props(5)
    by simp
  have Φ {is a filter on} (X×X)
  proof -
    from assms have ∅∉Φ
      unfolding Supersets_def using uniformity_base_props(2)
      by blast
    moreover have X×X ∈ Φ
    proof -
      from assms(2) obtain B where B∈𝔅
        using uniformity_base_props(6) by blast
      with ‹𝔅⊆Pow(X×X)› show X×X ∈ Φ unfolding Supersets_def
        by blast
    qed
    moreover have Φ ⊆ Pow(X×X) unfolding Supersets_def by auto
    moreover have ∀U∈Φ. ∀V∈Φ. U∩V ∈ Φ
    proof -
      { fix U V assume U∈Φ V∈Φ
        then obtain B1 B2 where B1∈𝔅 B2∈𝔅 B1⊆U B2⊆V
          unfolding Supersets_def by auto
        from assms(2) ‹B1∈𝔅› ‹B2∈𝔅› obtain B3 where B3∈𝔅 and B3⊆B1∩B2
          using uniformity_base_props(1) by blast
        from ‹B1⊆U› ‹B2⊆V› ‹B3⊆B1∩B2› have B3⊆U∩V by auto
        with ‹U∈Φ› ‹V∈Φ› ‹B3∈𝔅› have U∩V ∈ Φ
          unfolding Supersets_def by auto
      } thus thesis by simp
    qed
    moreover have ∀U∈Φ. ∀C∈Pow(X×X). U⊆C ⟶ C∈Φ
    proof -
      { fix U C assume U∈Φ C∈Pow(X×X) U⊆C
        from ‹U∈Φ› obtain B where B∈𝔅 and B⊆U
          unfolding Supersets_def by auto
        with ‹U⊆C› ‹C∈Pow(X×X)› have C∈Φ
          unfolding Supersets_def by auto
      } thus thesis by auto
    qed
    ultimately show Φ {is a filter on} (X×X)
      unfolding IsFilter_def by simp
  qed
  moreover have ∀U∈Φ. id(X)⊆U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U)∈Φ
  proof -
```

```
    { fix U assume U∈Φ
      then obtain B where B∈𝔅 and B⊆U
        unfolding Supersets_def by auto
      with assms(2) have id(X) ⊆ U
        using uniformity_base_props(2) by blast
      moreover
      from assms(2) ‹B∈𝔅› obtain V where V∈𝔅 and V O V ⊆ B
        using uniformity_base_props(4) by blast
      with ‹𝔅⊆Pow(X×X)› have V∈Φ using superset_gen by auto
      with ‹V O V ⊆ B› ‹B⊆U› have ∃V∈Φ. V O V ⊆ U by blast
      moreover
      from assms(2) ‹B∈𝔅› ‹B⊆U› obtain W where W∈𝔅 and W⊆converse(U)
        using uniformity_base_props(3) by blast
      with ‹U∈Φ› have converse(U) ∈ Φ unfolding Supersets_def
        by auto
      ultimately have id(X)⊆U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U)∈Φ
        by simp
    } thus thesis by simp
  qed
  ultimately show thesis unfolding IsUniformity_def by simp
qed
```

The assumption that X is not empty in `uniformity_base_is_base` above is neccessary as the assertion is false if X is empty.

```
lemma uniform_space_empty: assumes 𝔅 {is a uniform base on} ∅
  shows ¬(Supersets(∅×∅,𝔅) {is a uniformity on} ∅)
proof -
  { let Φ = Supersets(∅×∅,𝔅)
    assume Φ {is a uniformity on} ∅
    from assms have 𝔅={∅} using uniformity_base_props(5,6) by force
    with ‹Φ {is a uniformity on} ∅› have False
      using supersets_in_empty unif_filter unfolding IsFilter_def by auto
  } thus thesis by auto
qed
```

69.3 Least upper bound of a set of uniformities

Uniformities on a set X are naturally ordered by the inclusion relation. Specifically, for two uniformities $\mathcal{U}_1$ and $\mathcal{U}_2$ on a set X if $\mathcal{U}_1 \subseteq \mathcal{U}_2$ we say that $\mathcal{U}_2$ is finer than $\mathcal{U}_1$ or that $\mathcal{U}_1$ is coarser than $\mathcal{U}_2$. Turns out this order is complete: every nonempty set of uniformities has a least upper bound, i.e. a supremum.

We define `Uniformities(X)` as the set of all uniformities on X.

```
definition
  Uniformities(X) ≡ {Φ ∈ Pow(Pow(X×X)). Φ {is a uniformity on} X}
```

If Φ is a uniformity on X, then Φ is a collection of subsets of $X \times X$, hence it's a member of `Uniformities(X)`.

```
lemma unif_in_unifs: assumes Φ {is a uniformity on} X
  shows Φ ∈ Uniformities(X)
  using assms unfolding Uniformities_def IsUniformity_def IsFilter_def
  by auto
```

For nonempty sets the set of uniformities is not empty as well.

```
lemma unifomities_exist: assumes X≠∅ shows Uniformities(X)≠∅
  unfolding Uniformities_def using assms min_uniformity by auto
```

Uniformities on a set X are naturally ordered by inclusion, we call the resulting order relation `OrderOnUniformities`.

```
definition
  OrderOnUniformities(X) ≡ InclusionOn(Uniformities(X))
```

The order defined by inclusion on uniformities is a partial order.

```
lemma ord_unif_partial_ord:
  shows IsPartOrder(Uniformities(X),OrderOnUniformities(X))
  unfolding OrderOnUniformities_def using incl_is_partorder by simp
```

In particular, the order defined by inclusion on uniformities is antisymmetric. Having this as a separate fact is handy as we reference some lemmas proven for antisymmetric (not necessarily partial order) relations.

```
lemma ord_unif_antisymm: shows antisym(OrderOnUniformities(X))
  using ord_unif_partial_ord unfolding IsPartOrder_def by simp
```

If X is not empty then the singleton $\{X \times X\}$ is the minimal element of the set of uniformities on X ordered by inclusion and the collection of subsets of $X \times X$ that contain the diagonal is the maximal element.

```
theorem uniformities_min_max: assumes X≠∅ shows
  HasAminimum(OrderOnUniformities(X),Uniformities(X))
  Minimum(OrderOnUniformities(X),Uniformities(X)) = {X×X}
  HasAmaximum(OrderOnUniformities(X),Uniformities(X))
  Maximum(OrderOnUniformities(X),Uniformities(X)) = {U∈Pow(X×X). id(X)⊆U}
proof -
  let 𝔘 = Uniformities(X)
  let r = OrderOnUniformities(X)
  let M = {U∈Pow(X×X). id(X)⊆U}
  from assms have {X×X} ∈ 𝔘 and M∈𝔘
    unfolding Uniformities_def using min_uniformity max_uniformity
    by auto
  { fix Φ assume Φ ∈ 𝔘
    then have Φ {is a filter on} (X×X)
      unfolding Uniformities_def using unif_filter by simp
    with ‹{X×X}∈𝔘› ‹Φ∈𝔘› have ⟨{X×X},Φ⟩ ∈ r
      unfolding IsFilter_def OrderOnUniformities_def InclusionOn_def
      by simp
  } with  ‹{X×X} ∈ 𝔘› show HasAminimum(r,𝔘) and Minimum(r,𝔘) = {X×X}
```

```
    unfolding HasAminimum_def using Order_ZF_4_L15 ord_unif_antisymm
    by auto
  { fix Φ assume Φ ∈ 𝔘
    then have Φ ⊆ M unfolding IsUniformity_def Uniformities_def
      by auto
    with ‹M∈𝔘› ‹Φ∈𝔘›  have ⟨Φ,M⟩ ∈ r
      unfolding OrderOnUniformities_def InclusionOn_def by simp
  } with  ‹M∈𝔘› show HasAmaximum(r,𝔘) and Maximum(r,𝔘) = M
    unfolding HasAmaximum_def using Order_ZF_4_L14 ord_unif_antisymm
    by auto
qed
```

Given a set of uniformities $\mathcal{U}$ on X we define a collection of subsets of X called LUB_UnifBase (the least upper bound base in comments) as the set of all products of nonempty finite subsets of $\bigcup\mathcal{U}$. The "least upper bound base" term is not justified at this point, but we will show later that this set is actually a uniform base (i.e. a fundamental system of entourages) on X and hence the supersets of it form a uniformity on X, which is the supremum (i.e. the least upper bound) of $\mathcal{U}$.

```
definition LUB_UnifBase(𝒰) = {⋂M. M ∈ FinPow(⋃𝒰)\{∅}}
```

For any two sets in the least upper bound base there is a third one contained in both.

```
lemma lub_unif_base_1st_cond:
  assumes 𝒰⊆Uniformities(X)  U₁ ∈ LUB_UnifBase(𝒰)  U₂ ∈ LUB_UnifBase(𝒰)
  shows ∃U₃∈LUB_UnifBase(𝒰). U₃⊆U₁∩U₂
proof -
  let ℱ = FinPow(⋃𝒰)\{∅}
  from assms(2,3) obtain M₁ M₂ where
    M₁∈ℱ M₁≠∅ U₁=⋂M₁ M₂∈ℱ M₂≠∅ U₂=⋂M₂
    unfolding LUB_UnifBase_def by auto
  let M₃ = M₁∪M₂
  from ‹M₁≠∅› ‹M₂≠∅› ‹U₁=⋂M₁› ‹U₂=⋂M₂› have ⋂M₃ ⊆ U₁∩U₂
    by auto
  with ‹M₁ ∈ ℱ› ‹M₂ ∈ ℱ› ‹U₂=⋂M₂› show thesis
    using union_finpow unfolding LUB_UnifBase_def by auto
qed
```

Each set in the least upper bound base contains the diagonal of $X \times X$.

```
lemma lub_unif_base_2nd_cond:
  assumes 𝒰⊆Uniformities(X) U ∈ LUB_UnifBase(𝒰)
  shows id(X)⊆U
  using assms
  unfolding LUB_UnifBase_def FinPow_def Uniformities_def IsUniformity_def
  by blast
```

The converse of each set from the least upper bound base contains a set from it.

```
lemma lub_unif_base_3rd_cond:
  assumes 𝒰⊆Uniformities(X) U₁ ∈ LUB_UnifBase(𝒰)
  shows ∃U₂ ∈ LUB_UnifBase(𝒰). U₂ ⊆ converse(U₁)
proof -
  let ℱ = FinPow(⋃𝒰)\{∅}
  from assms(2) obtain M₁ where  M₁∈ℱ M₁≠∅ U₁=⋂M₁
    unfolding LUB_UnifBase_def by auto
  let M₂ = {converse(V). V∈M₁}
  from assms(1) ‹M₁∈ℱ› have ∀V∈M₁. converse(V) ∈ ⋃𝒰
    unfolding FinPow_def Uniformities_def using entourage_props(4)
    by blast
  with ‹M₁∈ℱ› have ⋂M₂ ∈ LUB_UnifBase(𝒰)
    using fin_image_fin0 unfolding LUB_UnifBase_def by auto
  from assms(1) ‹M₁∈ℱ› ‹U₁=⋂M₁› have ⋂M₂ ⊆ converse(U₁)
    unfolding Uniformities_def FinPow_def using prod_converse
    by blast
  with ‹⋂M₂ ∈ LUB_UnifBase(𝒰)› show thesis by auto
qed
```

For each set (relation) U_1 from the least upper bound base there is another one U_2 such that U_2 composed with itself is contained in U_1.

```
lemma lub_unif_base_4th_cond:
  assumes 𝒰⊆Uniformities(X) U₁ ∈ LUB_UnifBase(𝒰)
  shows ∃U₂ ∈ LUB_UnifBase(𝒰). U₂ O U₂ ⊆ U₁
proof -
  let ℱ = FinPow(⋃𝒰)\{∅}
  from assms(2) obtain M₁ where  M₁∈ℱ M₁≠∅ U₁=⋂M₁
    unfolding LUB_UnifBase_def by auto
  from ‹M₁∈ℱ› have Finite(M₁) unfolding FinPow_def by simp
  { fix V assume V∈M₁
    with assms(1) ‹M₁∈ℱ› obtain Φ where Φ∈𝒰 and V∈Φ
      unfolding FinPow_def by auto
    with assms(1) ‹V∈Φ› obtain W where W∈Φ and W O W ⊆ V
      unfolding Uniformities_def using entourage_props(3) by blast
    with ‹Φ∈𝒰› have ∃W∈⋃𝒰. W O W ⊆ V by auto
  } hence ∀V∈M₁. ∃W∈⋃𝒰. W O W ⊆ V by simp
  with ‹Finite(M₁)› have ∃f∈M₁→⋃𝒰. ∀V∈M₁. f(V) O f(V) ⊆ V
    by (rule finite_choice_fun)
  then obtain f where f:M₁→⋃𝒰 and ∀V∈M₁. f(V) O f(V) ⊆ V
    by auto
  let M₂ = {f(V). V∈M₁}
  from ‹f:M₁→⋃𝒰› have ∀V∈M₁. f(V) ∈ ⋃𝒰 using apply_funtype by blast
  with ‹M₁∈ℱ› have ⋂M₂ ∈ LUB_UnifBase(𝒰)
    using fin_image_fin0 unfolding LUB_UnifBase_def by auto
  from ‹M₁≠∅› ‹∀V∈M₁. f(V) O f(V) ⊆ V› have
    (⋂V∈M₁. f(V)) O (⋂V∈M₁. f(V)) ⊆ (⋂V∈M₁. V)
    by (rule square_incl_product)
  with ‹U₁=⋂M₁› ‹⋂M₂ ∈ LUB_UnifBase(𝒰)› show thesis by auto
qed
```

The least upper bound base is a collection of relations on X.

```
lemma lub_unif_base_5th_cond:
  assumes 𝒰⊆Uniformities(X) shows LUB_UnifBase(𝒰) ⊆ Pow(X×X)
  using assms unfolding Uniformities_def FinPow_def LUB_UnifBase_def
  by blast
```

If a collection of uniformities is nonempty, then the least upper bound base is non-empty as well.

```
lemma lub_unif_base_6th_cond: assumes 𝒰⊆Uniformities(X) 𝒰≠∅
  shows LUB_UnifBase(𝒰) ≠ ∅
proof -
  from assms(2) obtain Φ where Φ∈𝒰 by auto
  with assms(1) have ⋃𝒰 ≠ ∅ unfolding Uniformities_def
    using uniformity_non_empty by blast
  then show LUB_UnifBase(𝒰) ≠ ∅ using finpow_nempty_nempty
    unfolding LUB_UnifBase_def by simp
qed
```

If a collection of uniformities $\mathcal{U}$ is nonempty, $\mathcal{B}$ denotes the least upper bound base for $\mathcal{U}$, then $\mathcal{B}$ is a uniform base on X, hence its supersets form a uniformity on X and the uniform topology generated by that uniformity is indeed a topology on X.

```
theorem lub_unif_base_base:
  assumes X≠∅ 𝒰⊆Uniformities(X) 𝒰≠∅
  defines 𝔅 ≡ LUB_UnifBase(𝒰)
  shows
    𝔅 {is a uniform base on} X
    Supersets(X×X,𝔅) {is a uniformity on} X
    UniformTopology(Supersets(X×X,𝔅),X) {is a topology}
    ⋃UniformTopology(Supersets(X×X,𝔅),X) = X
  using assms lub_unif_base_1st_cond lub_unif_base_2nd_cond
    lub_unif_base_3rd_cond lub_unif_base_4th_cond lub_unif_base_5th_cond

    lub_unif_base_6th_cond uniformity_base_is_base uniform_top_is_top
  unfolding IsUniformityBaseOn_def by simp_all
```

At this point we know that supersets with respect to $X \times X$ of the least upper bound base for a collection of uniformities $\mathcal{U}$ form a uniformity. To shorten the notation we will call this uniformity `LUB_Unif(X,𝒰)`.

```
definition
  LUB_Unif(X,𝒰) ≡ Supersets(X×X,LUB_UnifBase(𝒰))
```

For any collection of uniformities $\mathcal{U}$ on a nonempty set X the `LUB_Unif(X,𝒰)` collection defined above is indeed an upper bound of $\mathcal{U}$ in the order defined by the inclusion relation.

```
lemma lub_unif_upper_bound:
  assumes X≠∅ 𝒰⊆Uniformities(X) Φ∈𝒰
```

```
  shows ⟨Φ,LUB_Unif(X,𝒰)⟩ ∈ OrderOnUniformities(X)
proof -
  let Ψ = LUB_Unif(X,𝒰)
  from assms have Ψ ∈ Uniformities(X)
    unfolding LUB_Unif_def using lub_unif_base_base(2) unif_in_unifs
    by blast
  from assms(2,3) have
    Φ ∈ Uniformities(X) and Φ {is a uniformity on} X
    unfolding Uniformities_def by auto
  { fix E assume E∈Φ
    with assms(3) have E ∈ LUB_UnifBase(𝒰)
      using singleton_in_finpow unfolding LUB_UnifBase_def
      by blast
    with ‹Φ {is a uniformity on} X› ‹E∈Φ› have E ∈ Ψ
      using entourage_props(1) superset_gen unfolding LUB_Unif_def
      by simp
  } hence Φ ⊆ Ψ by auto
  with ‹Φ ∈ Uniformities(X)› ‹Ψ ∈ Uniformities(X)› show thesis
    unfolding OrderOnUniformities_def InclusionOn_def by simp
qed
```

Any upper bound (in the order defined by inclusion relation) of a nonempty collection of uniformities $\mathcal{U}$ on a nonempty set X is greater or equal (in that order) than `LUB_Unif(X,𝒰)`. Together with `lub_unif_upper_bound` it means that `LUB_Unif(X,𝒰)` is indeed the least upper bound of $\mathcal{U}$.

```
lemma lub_unif_lub:
  assumes X≠∅  𝒰⊆Uniformities(X)  𝒰≠∅  and
    ∀Φ∈𝒰. ⟨Φ,Ψ⟩ ∈ OrderOnUniformities(X)
  shows ⟨LUB_Unif(X,𝒰),Ψ⟩ ∈ OrderOnUniformities(X)
proof -
  from assms(3,4) have Ψ ∈ Uniformities(X)
    unfolding OrderOnUniformities_def InclusionOn_def by auto
  then have Ψ {is a filter on} (X×X)
    unfolding Uniformities_def IsUniformity_def by simp
  from assms(4) have FinPow(⋃𝒰)\{∅} ⊆ FinPow(Ψ)\{∅}
    unfolding OrderOnUniformities_def InclusionOn_def FinPow_def
    by auto
  with ‹Ψ {is a filter on} (X×X)› have LUB_UnifBase(𝒰) ⊆ Ψ
    using filter_fin_inter_closed unfolding LUB_UnifBase_def by auto
  with ‹Ψ {is a filter on} (X×X)› have LUB_Unif(X,𝒰) ⊆ Ψ
    using filter_superset_closed unfolding LUB_Unif_def by simp
  with assms(1,2,3) ‹Ψ ∈ Uniformities(X)› show thesis
    using lub_unif_base_base(2) unif_in_unifs
    unfolding LUB_Unif_def OrderOnUniformities_def InclusionOn_def
    by simp
qed
```

A nonempty collection $\mathcal{U}$ of uniformities on X has a supremum (i.e. the least upper bound).

```
lemma lub_unif_sup: assumes X≠∅ 𝒰⊆Uniformities(X) 𝒰≠∅
  shows HasAsupremum(OrderOnUniformities(X),𝒰) and
    LUB_Unif(X,𝒰) = Supremum(OrderOnUniformities(X),𝒰)
proof -
  let r = OrderOnUniformities(X)
  let S = LUB_Unif(X,𝒰)
  from assms(1,2) have antisym(r) and ∀Φ∈𝒰. ⟨Φ,S⟩ ∈ r
    using ord_unif_antisymm lub_unif_upper_bound by simp_all
  from assms have I: ∀Ψ. (∀Φ∈𝒰. ⟨Φ,Ψ⟩ ∈ r) ⟶ ⟨S,Ψ⟩ ∈ r
    using lub_unif_lub by simp
  with assms(3) ‹antisym(r)› ‹∀Φ∈𝒰. ⟨Φ,S⟩ ∈ r› show HasAsupremum(r,𝒰)

    unfolding HasAsupremum_def using Order_ZF_5_L5(1) by blast
  from assms(3) ‹antisym(r)› ‹∀Φ∈𝒰. ⟨Φ,S⟩ ∈ r› I
  show S = Supremum(r,𝒰) using Order_ZF_5_L5(2) by blast
qed
```

The order on uniformities derived from inclusion is complete.

```
theorem uniformities_complete: assumes X≠∅
  shows OrderOnUniformities(X) {is complete}
proof -
  let r = OrderOnUniformities(X)
  { fix 𝒰 assume 𝒰≠∅ and IsBoundedAbove(𝒰,r)
    then obtain Ψ where ∀Φ∈𝒰. ⟨Φ,Ψ⟩ ∈ r
      unfolding IsBoundedAbove_def by auto
    then have 𝒰⊆Uniformities(X)
      unfolding OrderOnUniformities_def InclusionOn_def by auto
    with assms ‹𝒰≠∅› have HasAsupremum(r,𝒰)
      using lub_unif_sup by simp
  }
  then show r {is complete} unfolding HasAsupremum_def IsComplete_def
    by simp
qed

end
```

70 Metric spaces

```
theory MetricSpace_ZF imports Topology_ZF_1 OrderedLoop_ZF Lattice_ZF
UniformSpace_ZF
begin
```

A metric space is a set on which a distance between points is defined as a function $d : X \times X \to [0, \infty)$. With this definition each metric space is a topological space which is paracompact and Hausdorff (T_2), hence normal (in fact even perfectly normal).

70.1 Pseudometric - definition and basic properties

A metric on X is usually defined as a function $d : X \times X \rightarrow [0, \infty)$ that satisfies the conditions $d(x, x) = 0$, $d(x, y) = 0 \Rightarrow x = y$ (identity of indiscernibles), $d(x, y) = d(y, x)$ (symmetry) and $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality) for all $x, y \in X$. Here we are going to be a bit more general and define metric and pseudo-metric as a function valued in an ordered loop.

First we define a pseudo-metric, which has the axioms of a metric, but without the second part of the identity of indiscernibles. In our definition `IsApseudoMetric` is a predicate on five sets: the function d, the set X on which the metric is defined, the loop carrier G, the loop operation A and the order r on G.

definition
```
  IsApseudoMetric(d,X,G,A,r) ≡ d:X×X → Nonnegative(G,A,r)
    ∧ (∀x∈X. d⟨x,x⟩ = TheNeutralElement(G,A))
    ∧ (∀x∈X.∀y∈X. d⟨x,y⟩ = d⟨y,x⟩)
    ∧ (∀x∈X.∀y∈X.∀z∈X. ⟨d⟨x,z⟩, A⟨d⟨x,y⟩,d⟨y,z⟩⟩⟩ ∈ r)
```

We add the full axiom of identity of indiscernibles to the definition of a pseudometric to get the definition of metric.

definition
```
  IsAmetric(d,X,G,A,r) ≡
    IsApseudoMetric(d,X,G,A,r) ∧ (∀x∈X.∀y∈X. d⟨x,y⟩ = TheNeutralElement(G,A)
⟶ x=y)
```

A disk is defined as set of points located less than the radius from the center.

definition `Disk(X,d,r,c,R) ≡ {x∈X. ⟨d⟨c,x⟩,R⟩ ∈ StrictVersion(r)}`

We define `metric topology` as consisting of unions of open disks. Note the same definition is used for the topology generated by a pseudometric.

definition
```
  MetricTopology(X,L,A,r,d) ≡ {⋃𝒜. 𝒜 ∈ Pow(⋃c∈X. {Disk(X,d,r,c,R).
R∈PositiveSet(L,A,r)})}
```

Next we define notation for metric spaces. We will reuse the additive notation defined in the `loop1` locale adding only the assumption about d being a pseudometric and notation for a disk centered at c with radius R. Since for many theorems it is sufficient to assume the pseudometric axioms we will assume in this context that the sets d, X, L, A, r form a pseudometric raher than a metric. In the `pmetric_space` context τ denotes the topology defined by the metric d. Analogously to the notation defined in the `topology0` context `int(A)`, `cl(A)`, `∂A` will denote the interior, closure and boundary of the set A with respect to the metric topology.

locale `pmetric_space =  loop1 +`

```
  fixes d and X
  assumes pmetricAssum: IsApseudoMetric(d,X,L,A,r)
  fixes disk
  defines disk_def [simp]: disk(c,R) ≡ Disk(X,d,r,c,R)
  fixes pmettop (τ)
  defines pmettop [simp]: τ ≡ MetricTopology(X,L,A,r,d)
  fixes interior (int)
  defines interior_def [simp]: int(D) ≡ Interior(D,τ)
  fixes cl
  defines cl_def [simp]: cl(D) ≡ Closure(D,τ)
```

The next lemma shows the definition of the pseudometric in the notation used in the pmetric_space context.

```
lemma (in pmetric_space) pmetric_properties: shows
  d: X×X → L+
  ∀x∈X. d⟨x,x⟩ = 0
  ∀x∈X.∀y∈X. d⟨x,y⟩ = d⟨y,x⟩
  ∀x∈X.∀y∈X.∀z∈X. d⟨x,z⟩ ≤ d⟨x,y⟩ + d⟨y,z⟩
  using pmetricAssum unfolding IsApseudoMetric_def by auto
```

The values of the metric are in the in the nonnegative set of the loop, hence in the loop.

```
lemma (in pmetric_space) pmetric_loop_valued: assumes x∈X y∈X
  shows d⟨x,y⟩ ∈ L+ d⟨x,y⟩ ∈ L
proof -
  from assms show d⟨x,y⟩ ∈ L+ using pmetric_properties(1) apply_funtype
    by simp
  then show d⟨x,y⟩ ∈ L using Nonnegative_def by auto
qed
```

The definition of the disk in the notation used in the pmetric_space context:

```
lemma (in pmetric_space) disk_definition: shows disk(c,R) = {x∈X. d⟨c,x⟩
< R}
proof -
  have disk(c,R) = Disk(X,d,r,c,R) by simp
  then have disk(c,R) = {x∈X. ⟨d⟨c,x⟩,R⟩ ∈ StrictVersion(r)}
    unfolding Disk_def by simp
  moreover have ∀x∈X. ⟨d⟨c,x⟩,R⟩ ∈ StrictVersion(r) ⟷ d⟨c,x⟩ < R
    using def_of_strict_ver by simp
  ultimately show thesis by auto
qed
```

If the radius is positive then the center is in disk.

```
lemma (in pmetric_space) center_in_disk: assumes c∈X and R∈L+ shows
c ∈ disk(c,R)
  using pmetricAssum assms IsApseudoMetric_def PositiveSet_def disk_definition
by simp
```

A technical lemma that allows us to shorten some proofs: if c is an element of X and x is in disk with center c and radius R then R is a positive element of L and $-d(x,y)+R$ is in the set of positive elements of the loop.

```
lemma (in pmetric_space) radius_in_loop: assumes c∈X and x ∈ disk(c,R)
  shows R∈L 0<R R∈L₊ (-d⟨c,x⟩ + R) ∈ L₊
proof -
  from assms(2) have x∈X and d⟨c,x⟩ < R using disk_definition by auto
  with assms(1) show 0<R using pmetric_properties(1) apply_funtype
      nonneg_definition loop_strict_ord_trans by blast
  then show R∈L and R∈L₊ using posset_definition PositiveSet_def by
auto
  from ‹d⟨c,x⟩ < R› show (-d⟨c,x⟩ + R) ∈ L₊
    using ls_other_side(2) by simp
qed
```

If a point x is inside a disk B and $m \leq -d\langle c,x\rangle + R$ then the disk centered at the point x and with radius m is contained in the disk B.

```
lemma (in pmetric_space) disk_in_disk:
  assumes c∈X  and x ∈ disk(c,R) and m ≤ (-d⟨c,x⟩ + R)
  shows disk(x,m) ⊆ disk(c,R)
proof
  fix y assume y ∈ disk(x,m)
  then have d⟨x,y⟩ < m using disk_definition by simp
  from assms(1,2) ‹y ∈ disk(x,m)› have R∈L x∈X y∈X
    using radius_in_loop(1) disk_definition by auto
  with assms(1) have d⟨c,y⟩ ≤ d⟨c,x⟩ + d⟨x,y⟩ using pmetric_properties(4)
by simp
  from assms(1) ‹x∈X› have d⟨c,x⟩ ∈ L
    using pmetric_properties(1) apply_funtype nonneg_subset by auto
  with ‹d⟨x,y⟩ < m› assms(3) have d⟨c,x⟩ + d⟨x,y⟩ < d⟨c,x⟩ + (-d⟨c,x⟩ +
R)
    using loop_strict_ord_trans1 strict_ord_trans_inv(2) by blast
  with ‹d⟨c,x⟩ ∈ L› ‹R∈L› ‹d⟨c,y⟩ ≤ d⟨c,x⟩ + d⟨x,y⟩› ‹y∈X› show y ∈ disk(c,R)
    using lrdiv_props(6) loop_strict_ord_trans disk_definition by simp
qed
```

A special case of `disk_in_disk` where we set $m = -d\langle c,x\rangle + R$: if x is an element of a disk with center $c \in X$ and radius R then this disk contains the disk centered at x and with radius $-d\langle c,x\rangle + R$.

```
lemma (in pmetric_space) disk_in_disk1:
  assumes c∈X  and x ∈ disk(c,R)
  shows disk(x,-d⟨c,x⟩ + R) ⊆ disk(c,R)
proof -
  from assms(2) have R∈L and d⟨c,x⟩ ∈ L
    using disk_definition less_members by auto
  with assms show thesis using left_right_sub_closed(1) loop_ord_refl
disk_in_disk
    by simp
```

qed

Assuming that two disks have the same center, closed disk with smaller radius in contained in the (open) disk with a larger radius.

```
lemma (in pmetric_space) disk_radius_strict_mono:
  assumes r₁ < r₂
  shows {y∈X. d⟨x,y⟩ ≤ r₁} ⊆ disk(x,r₂)
  using assms loop_strict_ord_trans disk_definition by auto
```

If we assume that the loop's order relation down-directs L_+ then the collection of disks centered at points of the space and with radii in the positive set of the loop satisfies the base condition. The property that an order relation "down-directs" a set is defined in `Order_ZF` and means that every two-element subset of the set has a lower bound in that set.

```
lemma (in pmetric_space) disks_form_base:
  assumes r {down-directs} L₊
  defines B ≡ ⋃c∈X. {disk(c,R). R∈L₊}
  shows B {satisfies the base condition}
proof -
  { fix U V assume U∈B V∈B
    fix x assume x∈U∩V
    have ∃W∈B. x∈W ∧ W⊆U∩V
    proof -
      from assms(2) ‹U∈B› ‹V∈B› obtain c_U  c_V R_U  R_V
        where c_U ∈ X R_U ∈ L₊ c_V ∈ X R_V ∈ L₊ U = disk(c_U,R_U) V = disk(c_V,R_V)
        by auto
      with ‹x∈U∩V› have x ∈ disk(c_U,R_U) and x ∈ disk(c_V,R_V) by auto
      then have x∈X d⟨c_U,x⟩ < R_U d⟨c_V,x⟩ < R_V using disk_definition by
auto
      let m_U = - d⟨c_U,x⟩ + R_U
      let m_V = - d⟨c_V,x⟩ + R_V
      from ‹c_U∈X› ‹x∈disk(c_U,R_U)› ‹c_V∈X› ‹x∈disk(c_V,R_V)› have m_U∈L₊
and m_V∈L₊
        using radius_in_loop(4) by auto
      with assms(1) obtain m where m ∈ L₊ m ≤ m_U m ≤ m_V
        unfolding DownDirects_def by auto
      let W = disk(x,m)
      from ‹m ∈ L₊› ‹m ≤ m_U› ‹m ≤ m_V›
        ‹c_U ∈ X› ‹x ∈ disk(c_U,R_U)› ‹c_V ∈ X› ‹x ∈ disk(c_V,R_V)›
        ‹U = disk(c_U,R_U)› ‹V = disk(c_V,R_V)›
        have W ⊆ U∩V using disk_in_disk by blast
      moreover from assms(2) ‹x∈X› ‹m ∈ L₊› have W ∈ B and x∈W us-
ing center_in_disk
        by auto
      ultimately show thesis by auto
    qed
  } then show thesis unfolding SatisfiesBaseCondition_def by auto
qed
```

Disks centered at points farther away than the sum of radii do not overlap.

```
lemma (in pmetric_space) far_disks:
  assumes x∈X y∈X  rₓ+r_y ≤ d⟨x,y⟩
  shows disk(x,rₓ)∩disk(y,r_y) = ∅
proof -
  { assume disk(x,rₓ)∩disk(y,r_y) ≠ ∅
    then obtain z where z ∈ disk(x,rₓ)∩disk(y,r_y) by auto
    then have z∈X and d⟨x,z⟩ + d⟨y,z⟩ < rₓ+r_y
      using disk_definition add_ineq_strict by auto
    moreover from assms(1,2) ‹z∈X› have d⟨x,y⟩ ≤ d⟨x,z⟩ + d⟨y,z⟩
      using pmetric_properties(3,4) by auto
    ultimately have d⟨x,y⟩ < rₓ+r_y using loop_strict_ord_trans
      by simp
    with assms(3) have False using loop_strict_ord_trans by auto
  } thus thesis by auto
qed
```

If we have a loop element that is smaller than the distance between two points, then we can separate these points with disks.

```
lemma (in pmetric_space) disjoint_disks:
  assumes x∈X y∈X rₓ<d⟨x,y⟩
  shows (-rₓ+(d⟨x,y⟩)) ∈ L₊ and disk(x,rₓ)∩disk(y,-rₓ+(d⟨x,y⟩)) = 0
proof -
  from assms(3) show (-rₓ+(d⟨x,y⟩)) ∈ L₊
    using ls_other_side posset_definition1 by simp
  from assms(1,2,3) have rₓ∈L and d⟨x,y⟩ ∈ L
    using less_members(1) pmetric_loop_valued(2) by auto
  then have rₓ+(-rₓ+(d⟨x,y⟩)) = d⟨x,y⟩ using lrdiv_props(6) by simp
  with assms(1,2) ‹d⟨x,y⟩ ∈ L› show disk(x,rₓ)∩disk(y,-rₓ+(d⟨x,y⟩)) =
0
    using loop_ord_refl far_disks by simp
qed
```

The definition of metric topology written in notation of `pmetric_space` context:

```
lemma (in pmetric_space) metric_top_def_alt:
  defines B ≡ ⋃c∈X. {disk(c,R). R∈L₊}
  shows τ = {⋃A. A ∈ Pow(B)}
proof -
  from assms have MetricTopology(X,L,A,r,d) =  {⋃A. A ∈ Pow(B)}
    unfolding MetricTopology_def by simp
  thus thesis by simp
qed
```

If the order of the loop down-directs its set of positive elements then the metric topology defined as collection of unions of (open) disks is indeed a topology. Recall that in the `pmetric_space` context τ denotes the metric topology.

```
theorem (in pmetric_space) pmetric_is_top:
  assumes  r {down-directs} L₊
  shows τ {is a topology}
  using assms disks_form_base Top_1_2_T1 metric_top_def_alt by simp
```

If r down-directs L_+ then the collection of open disks is a base for the metric topology.

```
theorem (in pmetric_space) disks_are_base:
  assumes  r {down-directs} L₊
  defines B ≡ ⋃c∈X. {disk(c,R). R∈L₊}
  shows B {is a base for} τ
  using assms disks_form_base Top_1_2_T1 metric_top_def_alt by simp
```

If r down-directs L_+ then X is the carrier of metric topology.

```
theorem (in pmetric_space) metric_top_carrier:
  assumes  r {down-directs} L₊ shows ⋃τ = X
proof -
  let B = ⋃c∈X. {disk(c,R). R∈L₊}
  from assms have ⋃τ = ⋃B
    using disks_are_base Top_1_2_L5 by simp
  moreover have ⋃B = X
  proof
    show ⋃B ⊆ X using disk_definition by auto
    from assms show X ⊆ ⋃B unfolding DownDirects_def using center_in_disk

      by blast
  qed
  ultimately show ⋃τ = X by simp
qed
```

Under the assumption that r down-directs L_+ the propositions proven in the `topology0` context can be used in the `pmetric_space` context.

```
lemma (in pmetric_space) topology0_valid_in_pmetric_space:
  assumes  r {down-directs} L₊
  shows topology0(τ)
  using assms pmetric_is_top unfolding topology0_def by simp
```

If r down-directs L_+ then disks are open in the metric topology.

```
lemma (in pmetric_space) disks_open:
  assumes c∈X R∈L₊ r {down-directs} L₊
  shows disk(c,R) ∈ τ
  using assms base_sets_open disks_are_base(1) pmetric_is_top
  by blast
```

If r down-directs L_+ and x is an element of an open set U then there exist radius $R \in L_+$ such that the disk with center x and radius R is contained in U.

```
lemma (in pmetric_space) point_open_disk:
```

```
  assumes r {down-directs} L₊ U∈τ x∈U
  shows ∃R∈L₊. disk(x,R) ⊆ U
proof -
  let B = ⋃c∈X. {disk(c,R). R∈L₊}
  from assms have ∃V∈B. V⊆U ∧ x∈V
    using disks_are_base point_open_base_neigh by force
  then obtain c R where c∈X R∈L₊ x∈disk(c,R) disk(c,R)⊆U
    by auto
  then show thesis using radius_in_loop(4) disk_in_disk1
    by blast
qed
```

If r down-directs L_+ then the generated topology cannot distinguish two points if their distance is zero. "Cannot distinguish" here means that if one is in an open set then the second one is in that set too.

```
lemma (in pmetric_space) zero_dist_same_open:
  assumes r {down-directs} L₊ U∈τ x∈U y∈X d⟨x,y⟩=0
  shows y∈U
  using assms point_open_disk posset_definition1 disk_definition
    by force
```

A pseudometric that induces a T_0 topology is a metric.

```
theorem (in pmetric_space) pmetric_t0_metric:
  assumes r {down-directs} L₊ and τ {is T₀}
  shows IsAmetric(d,X,L,A,r)
proof -
  { fix x y
    assume x∈X y∈X d⟨x,y⟩=0
    with assms(1) have ∀U∈τ. (x∈U ⟷ y∈U)
      using zero_dist_same_open pmetric_properties(3) by auto
    with assms ‹x∈X› ‹y∈X› have x=y using metric_top_carrier Top_1_1_L1
      by blast
  } then show IsAmetric(d,X,L,A,r)
    using pmetricAssum unfolding IsAmetric_def by auto
qed
```

To define the metric_space locale we take the pmetric_space and add the assumption of identity of indiscernibles.

```
locale metric_space =  pmetric_space +
  assumes ident_indisc: ∀x∈X. ∀y∈X. d⟨x,y⟩=0 ⟶ x=y
```

In the metric_space locale d is a metric.

```
lemma (in metric_space) d_metric: shows IsAmetric(d,X,L,A,r)
  using pmetricAssum ident_indisc unfolding IsAmetric_def by simp
```

Distance of different points is greater than zero.

```
lemma (in metric_space) dist_pos: assumes x∈X y∈X x≠y
  shows 0<d⟨x,y⟩ d⟨x,y⟩ ∈ L₊
```

```
proof -
  from assms(1,2) have d⟨x,y⟩ ∈ L⁺
    using pmetric_properties(1) apply_funtype by simp
  then have 0 ≤ d⟨x,y⟩ using Nonnegative_def by auto
  with assms show d⟨x,y⟩ ∈ L₊ and 0<d⟨x,y⟩
    using ident_indisc posset_definition posset_definition1 by auto
qed
```

If r down-directs L_+ then the ordered loop valued metric space is T_2 (i.e. Hausdorff).

```
theorem (in metric_space) metric_space_T2:
    assumes r {down-directs} L₊
    shows τ {is T₂}
proof -
  let B = ⋃c∈X. {disk(c,R). R∈L₊}
  { fix x y assume x∈⋃τ y∈⋃τ x≠y
    from assms have B⊆τ using metric_top_def_alt by auto
    have ∃U∈B. ∃V∈B. x∈U ∧ y∈V ∧ U∩V = ∅
    proof -
      let R = d⟨x,y⟩
      from assms have ⋃τ = X using metric_top_carrier by simp
      with ‹x∈⋃τ› have x∈X by blast
      from ‹⋃τ = X› ‹y∈⋃τ› have y∈X by blast
      with ‹x≠y› ‹x∈X› have R∈L₊ using dist_pos by simp
      with ‹x∈X› ‹y∈X› have disk(x,R) ∈ B and disk(y,R) ∈ B
        by auto
      { assume disk(x,R) ∩ disk(y,R) = ∅
        moreover from ‹x∈X› ‹y∈X› ‹R∈L₊› have
            disk(x,R)∈B disk(y,R)∈B x∈disk(x,R) y∈disk(y,R)
          using center_in_disk by auto
        ultimately have ∃U∈B. ∃V∈B. x∈U ∧ y∈V ∧ U∩V = 0 by blast
      }
      moreover
      { assume disk(x,R) ∩ disk(y,R) ≠ 0
        then obtain z where z ∈ disk(x,R) and z ∈ disk(y,R)
          by auto
        then have d⟨x,z⟩ < R using disk_definition by simp
        then have 0 < -d⟨x,z⟩+R using ls_other_side(1) by simp
        let r = -d⟨x,z⟩+R
        have r<R
        proof -
          from ‹z ∈ disk(y,R)› ‹x∈X› ‹y∈X› have z∈X x≠z
            using disk_definition pmetric_properties(3) by auto
          with ‹x∈X› ‹y∈X› ‹z∈X› show thesis
            using pmetric_loop_valued dist_pos(1) add_subtract_pos(2)
by simp
        qed
        with ‹x∈X› ‹y∈X› have disk(x,r)∩disk(y,-r+R) = 0
          by (rule disjoint_disks)
```

```
      moreover
      from ‹0<r› ‹r<R› have r∈L+ (-r+R) ∈ L+
        using ls_other_side posset_definition1 by auto
      with ‹x∈X› ‹y∈X› have
          disk(x,r)∈B disk(y,-r+(d⟨x,y⟩))∈B and
          x∈disk(x,r) y∈disk(y,-r+(d⟨x,y⟩))
        using center_in_disk by auto
      ultimately have ∃U∈B. ∃V∈B. x∈U ∧ y∈V ∧ U∩V = 0 by blast
    }
    ultimately show thesis by auto
  qed
  with ‹B⊆τ› have ∃U∈τ. ∃V∈τ. x∈U ∧ y∈V ∧ U∩V = ∅ by (rule exist2_subset)
} then show thesis unfolding isT2_def by simp
qed
```

70.2 Uniform structures on (pseudo-)metric spaces

Each pseudometric space with pseudometric $d : X \times X \to L^+$ supports a natural uniform structure, defined as supersets of the collection of inverse images $U_c = d^{-1}([0, c])$, where $c > 0$.

In the following definition X is the underlying space, L is the loop (carrier), A is the loop operation, r is an order relation compatible with A, and d is a pseudometric on X, valued in the ordered loop L. With this we define the uniform gauge as the collection of inverse images of the closed intervals $[0, c]$ as c varies of the set of positive elements of L. See uniform_gauge_def_alt for this definition in a more readable notation.

```
definition
  UniformGauge(X,L,A,r,d) ≡ {d-({c∈Nonnegative(L,A,r). ⟨c,b⟩ ∈ r}). b∈PositiveSet(L,A,r)}
```

In the pmetric_space context we will write UniformGauge(X,L,A,r,d) as 𝔅.

```
abbreviation (in pmetric_space) gauge (𝔅) where 𝔅 ≡ UniformGauge(X,L,A,r,d)
```

In notation defined in the pmetric_space context we can write the uniform gauge as $\{d^{-1}(\{c \in L^+ : c \leq b\}) : b \in L_+\}$.

```
lemma (in pmetric_space) uniform_gauge_def_alt:
  shows 𝔅 = {d-({c∈L+. c≤b}). b∈L+}
  unfolding UniformGauge_def by simp
```

Members of the uniform gauge are subsets of $X \times X$ i.e. relations on X.

```
lemma (in pmetric_space) uniform_gauge_relations:
  assumes B∈𝔅 shows B⊆X×X
  using assms uniform_gauge_def_alt pmetric_properties(1) func1_1_L3
  by force
```

If the distance between two points of X is less or equal b, then this pair of points is in $d^{-1}([0, b])$.

```
lemma (in pmetric_space) gauge_members:
  assumes x∈X y∈X d⟨x,y⟩ ≤ b
  shows ⟨x,y⟩ ∈ d-({c∈L⁺. c≤b})
  using assms pmetric_properties(1) apply_funtype func1_1_L15
  by simp
```

Suppose $b \in L_+$ (i.e. b is an element of the loop that is greater than the neutral element) and $x \in X$. Then the image of the singleton set $\{x\}$ by the relation $B = \{d^{-1}(\{c \in L^+ : c \leq b\}$ is the set $\{y \in X : d\langle x, y\rangle \leq b\}$, i.e. the closed disk with center x and radius b. Hence the the image $B\{x\}$ contains the open disk with center x and radius b.

```
lemma (in pmetric_space) disk_in_gauge:
  assumes b∈L₊ x∈X
  defines B ≡ d-({c∈L⁺. c≤b})
  shows B{x} = {y∈X. d⟨x,y⟩ ≤ b} and disk(x,b) ⊆ B{x}
proof -
  from assms(1,3) have B⊆X×X
    using uniform_gauge_def_alt uniform_gauge_relations by auto
  with assms(2,3) show B{x} = {y∈X. d⟨x,y⟩ ≤ b}
    using pmetric_properties(1) func1_1_L15 by force
  then show disk(x,b) ⊆ B{x} using disk_definition by auto
qed
```

Gauges corresponding to larger elements of the loop are larger.

```
lemma (in pmetric_space) uniform_gauge_mono:
  assumes b₁≤b₂ shows d-({c∈L⁺. c≤b₁}) ⊆ d-({c∈L⁺. c≤b₂})
  using ordLoopAssum assms vimage_mono1
  unfolding IsAnOrdLoop_def IsPartOrder_def trans_def by auto
```

For any two sets of the form $d^{-1}([0, b])$ we can find a third one that is contained in both.

```
lemma (in pmetric_space) gauge_1st_cond:
  assumes r {down-directs} L₊ B₁∈𝔅 B₂∈𝔅
  shows ∃B₃∈𝔅. B₃⊆B₁∩B₂
proof -
  from assms(2,3) obtain b₁ b₂ where b₁∈L₊ b₂∈L₊ and
    I: B₁ = d-({c∈L⁺. c≤b₁}) B₂ = d-({c∈L⁺. c≤b₂})
    using uniform_gauge_def_alt by auto
  from assms(1) ‹b₁∈L₊› ‹b₂∈L₊› obtain b₃ where b₃∈L₊ b₃≤b₁ b₃≤b₂
    unfolding DownDirects_def by auto
  from I ‹b₃≤b₁› ‹b₃≤b₂› have d-({c∈L⁺. c≤b₃}) ⊆ B₁∩B₂
    using uniform_gauge_mono by blast
  with ‹b₃∈L₊› show thesis using uniform_gauge_def_alt
    by auto
qed
```

Sets of the form $d^{-1}([0, b])$ contain the diagonal.

```
lemma (in pmetric_space) gauge_2nd_cond: assumes B∈𝔅 shows id(X)⊆B
```

```
proof
  fix p assume p∈id(X)
  then obtain x where x∈X and p=⟨x,x⟩ by auto
  then have p∈X×X and d(p) = 0 using pmetric_properties(2) by simp_all
  from assms obtain b where b∈L₊ and B = d-({c∈L⁺. c≤b})
    using uniform_gauge_def_alt by auto
  with ‹p∈X×X› ‹d(p) = 0› show p∈B
    using posset_definition1 loop_zero_nonneg pmetric_properties(1) func1_1_L15
    by simp
qed
```

Sets of the form $d^{-1}([0,b])$ are symmetric.

```
lemma (in pmetric_space) gauge_symmetric:
  assumes B∈𝔅 shows B = converse(B)
proof -
  from assms obtain b where B = d-({c∈L⁺. c≤b})
    using uniform_gauge_def_alt by auto
  with pmetricAssum show thesis unfolding IsApseudoMetric_def
    using symm_vimage_symm by auto
qed
```

A set of the form $d^{-1}([0,b])$ contains a symmetric set of this form.

```
corollary (in pmetric_space) gauge_3rd_cond:
  assumes B₁∈𝔅 shows ∃B₂∈𝔅. B₂ ⊆ converse(B₁)
  using assms gauge_symmetric by auto
```

The collection of sets of the form $d^{-1}([0,b])$ for $b \in L_+$ is contained of the powerset of $X \times X$.

```
lemma (in pmetric_space) gauge_5thCond: shows 𝔅⊆Pow(X×X)
  using uniform_gauge_def_alt pmetric_properties(1) func1_1_L3 by force
```

If the set of positive values is non-empty, then there are sets of the form $d^{-1}([0,b])$ for $b > 0$.

```
lemma (in pmetric_space) gauge_6thCond:
  assumes L₊≠∅ shows 𝔅≠∅  using assms uniform_gauge_def_alt by simp
```

The remaining 4th condition for the sets of the form $d^{-1}([0,b])$ to be a uniform base (or a fundamental system of entourages) cannot be proven without additional assumptions in the context of ordered loop valued metrics. To see that consider the example of natural numbers with the metric $d\langle x,y\rangle = |x-y|$, where we think of d as valued in the nonnegative set of ordered group of integers. Now take the set $B_1 = d^{-1}([0,1]) = d^{-1}(\{0,1\})$. Then the set $B_1 \circ B_1$ is strictly larger than B_1, but there is no smaller set B_2 we can take so that $B_2 \circ B_2 \subseteq B_1$. One condition that is sufficient is that for every $b_1 > 0$ there is a $b_2 > 0$ such that $b_2 + b_2 \le b_1$. I have not found a standard name for this property, for now we will use the name `IsHalfable`.

definition

```
  IsHalfable(L,A,r) ≡ ∀b₁∈PositiveSet(L,A,r). ∃b₂∈PositiveSet(L,A,r).
⟨A⟨b₂,b₂⟩,b₁⟩ ∈ r
```

The property of halfability written in the notation used in the `pmetric_space` context.

```
lemma (in pmetric_space) is_halfable_def_alt:
  assumes IsHalfable(L,A,r) b₁∈L₊
  shows ∃b₂∈L₊. b₂+b₂ ≤ b₁
  using assms unfolding IsHalfable_def by simp
```

If $B_i = d^{-1}(\{c \in L_+ : c \leq b_i\})$ for $i = 1, 2$ and $b_2+b_2 \leq b1$ then $B_2 \circ B_2 \leq B_1$. The proof uses the triangle inequality so it's not really a property of ordered loops only.

```
lemma (in pmetric_space) half_vimage_square:
  assumes b₂∈L₊ and b₂+b₂ ≤ b₁
  defines B₁ ≡ d-({c∈L⁺. c≤b₁}) and B₂ ≡ d-({c∈L⁺. c≤b₂})
  shows B₂ O B₂ ⊆ B₁
proof
  from assms(1,4) have B₂⊆X×X using pmetric_properties(1) func1_1_L3
by simp
  fix p assume p ∈ B₂ O B₂
  with ‹B₂⊆X×X› obtain x y where x∈X y∈X and p=⟨x,y⟩
    by blast
  from ‹p ∈ B₂ O B₂› ‹p=⟨x,y⟩› obtain z where ⟨x,z⟩ ∈ B₂ and ⟨z,y⟩ ∈
B₂
      using rel_compdef by auto
  with ‹B₂⊆X×X› have z∈X by auto
  from assms(4) ‹⟨x,z⟩ ∈ B₂› ‹⟨z,y⟩ ∈ B₂› have d⟨x,z⟩ + d⟨z,y⟩ ≤ b₂+ b₂
    using pmetric_properties(1) func1_1_L15 add_ineq by simp
  with ‹b₂+b₂ ≤ b₁› have d⟨x,z⟩ + d⟨z,y⟩ ≤ b₁
    using loop_ord_trans by simp
  with assms(3) ‹x∈X› ‹y∈X› ‹z∈X› ‹p=⟨x,y⟩› show p∈B₁
    using pmetric_properties(4) loop_ord_trans gauge_members by blast
qed
```

If the loop order is halfable then for every set B_1 of the form $d^{-1}([0, b_1])$ for some $b_1 > 0$ we can find another one $B_2 = d^{-1}([0, b_2])$ such that B_2 composed with itself is contained in B_1.

```
lemma (in pmetric_space) gauge_4thCond:
  assumes IsHalfable(L,A,r) B₁∈𝔅 shows ∃B₂∈𝔅. B₂ O B₂ ⊆ B₁
proof -
  from assms(2) obtain b₁ where b₁∈L₊ and B₁ = d-({c∈L⁺. c≤b₁})
    using uniform_gauge_def_alt by auto
  from assms(1) ‹b₁∈L₊› obtain b₂ where b₂∈L₊ and b₂+b₂ ≤ b₁
    using is_halfable_def_alt by auto
  with ‹B₁ = d-({c∈L⁺. c≤b₁})› show thesis
    using half_vimage_square unfolding UniformGauge_def by force
qed
```

If X and L_+ are not empty, the order relation r down-directs L_+, and the loop order is halfable, then $\mathfrak{B}$ (which in the pmetric_space context is an abbreviation for $\{d^{-1}(\{c \in L^+ : c \leq b\} : b \in L_+\})$ is a fundamental system of entourages, hence its supersets form a uniformity on X and hence those supersets define a topology on X.

```
theorem (in pmetric_space) metric_gauge_base:
  assumes X≠∅ L₊≠∅ r {down-directs} L₊ IsHalfable(L,A,r)
  shows
    𝔅 {is a uniform base on} X
    Supersets(X×X,𝔅) {is a uniformity on} X
    UniformTopology(Supersets(X×X,𝔅),X) {is a topology}
    ⋃UniformTopology(Supersets(X×X,𝔅),X) = X
  using assms gauge_1st_cond gauge_2nd_cond gauge_3rd_cond
    gauge_4thCond gauge_5thCond gauge_6thCond uniformity_base_is_base
    uniform_top_is_top
  unfolding IsUniformityBaseOn_def by simp_all
```

At this point we know that a pseudometric induces two topologies: one consisting of unions of open disks (the metric topology) and second one being the uniform topology derived from the uniformity generated the fundamental system of entourages (the base uniformity) of the sets of the form $d^{-1}([0,b])$ for $b > 0$. The next theorem states that if X and L_+ are not empty, r down-directs L_+, and the loop order is halfable, then these two topologies are in fact the same. Recall that in the pmetric_space context τ denotes the metric topology.

```
theorem (in pmetric_space) metric_top_is_uniform_top:
  assumes X≠∅ L₊≠∅ r {down-directs} L₊ IsHalfable(L,A,r)
  shows τ = UniformTopology(Supersets(X×X,𝔅),X)
proof
  let Φ = Supersets(X×X,𝔅)
  from assms have Φ {is a uniformity on} X using metric_gauge_base
    by simp
  let T = UniformTopology(Φ,X)
  { fix U assume U∈T
    then have U∈Pow(X) and I: ∀x∈U. U∈{V{x}. V∈Φ}
      unfolding UniformTopology_def by auto
    { fix x assume x∈U
      with I obtain A where A∈Φ and U = A{x}
        by auto
      from ‹x∈U› ‹U∈T› have x∈⋃T by auto
      with assms have x∈X using metric_gauge_base(4) by simp
      from ‹A∈Φ› obtain B where B∈𝔅 and B⊆A
        unfolding Supersets_def by auto
      from ‹B∈𝔅› obtain b where b∈L₊ and B = d-({c∈L⁺. c≤b})
        using uniform_gauge_def_alt by auto
      with ‹x∈X› ‹B⊆A› ‹U = A{x}› have disk(x,b) ⊆ U
        using disk_in_gauge(2) by blast
```

```
      with assms(3) ‹x∈X› ‹b∈L₊› have ∃V∈τ. x∈V ∧ V⊆U
        using disks_open center_in_disk by force
    } with assms(3) have U∈τ
      using topology0_valid_in_pmetric_space topology0.open_neigh_open
        by simp
  } thus T ⊆ τ by auto
  let 𝒟 = ⋃c∈X. {disk(c,R). R∈L₊}
  { fix U assume U ∈ 𝒟
    then obtain c b where c∈X b∈L₊ U = disk(c,b)
      by blast
    { fix x assume x∈U
      let b₁ = -d⟨c,x⟩ + b
      from ‹x∈U› ‹c∈X› ‹U = disk(c,b)› have
        x∈X x∈disk(c,b) disk(x,b₁) ⊆ U b₁ ∈ L₊
        using disk_in_disk1 disk_definition radius_in_loop(4) by simp_all
      with assms(4) obtain b₂ where b₂∈L₊ and b₂+b₂ ≤ b₁
        using is_halfable_def_alt by auto
      let D = {y∈X. d⟨x,y⟩ ≤ b₂}
      from ‹b₂∈L₊› ‹b₂+b₂ ≤ b₁› have D ⊆ disk(x,b₁)
        using posset_definition1 positive_subset add_subtract_pos(3)
          loop_strict_ord_trans1 disk_radius_strict_mono by blast
      let B = d-({c∈L⁺. c≤b₂})
      from ‹b₂∈L₊› have B∈𝔅 using uniform_gauge_def_alt by auto
      then have B∈Φ using uniform_gauge_relations superset_gen
        by simp
      from ‹b₂∈L₊› ‹x∈X› ‹D ⊆ disk(x,b₁)› ‹disk(x,b₁) ⊆ U› have B{x}
⊆ U
        using disk_in_gauge(1) by auto
      with ‹B∈Φ› have ∃W∈Φ. W{x} ⊆ U by auto
    } with ‹U = disk(c,b)› ‹Φ {is a uniformity on} X› have U ∈ T
      using disk_definition uniftop_def_alt1 by auto
  } hence 𝒟 ⊆ T by auto
  with assms show τ⊆T
    using disks_are_base(1) metric_gauge_base(3) base_smallest_top
    by simp
qed

end
```

71 Basic properties of real numbers

```
theory Real_ZF_2 imports OrderedField_ZF MetricSpace_ZF
begin
```

Isabelle/ZF and IsarMathLib do not have a set of real numbers built-in. The `Real_ZF` and `Real_ZF_1` theories provide a construction but here we do not use it in any way and we just assume that we have a model of real numbers (i.e. a completely ordered field) as defined in the `Ordered_Field` theory. The construction only assures us that objects with the desired properties exist in the ZF world.

71.1 Basic notation for real numbers

In this section we define notation that we will use whenever real numbers play a role, i.e. most of mathematics.

The next locale sets up notation for contexts where real numbers are used. Note we define the (real) natural numbers ℕ as starting from one.

```
locale reals =
  fixes Reals(ℝ) and Add and Mul and ROrd
  assumes R_are_reals: IsAmodelOfReals(ℝ,Add,Mul, ROrd)

  fixes zero (0)
  defines zero_def[simp]: 0 ≡ TheNeutralElement(ℝ,Add)

  fixes one (1)
  defines one_def[simp]: 1 ≡ TheNeutralElement(ℝ,Mul)

  fixes realmul (infixl · 71)
  defines realmul_def[simp]: x · y ≡ Mul⟨x,y⟩

  fixes realadd (infixl + 69)
  defines realadd_def[simp]: x + y ≡ Add⟨x,y⟩

  fixes realminus(- _ 89)
  defines realminus_def[simp]: (-x) ≡ GroupInv(ℝ,Add)(x)

  fixes realsub (infixl - 90)
  defines realsub_def [simp]: x-y ≡ x+(-y)

  fixes lesseq (infix ≤ 68)
  defines lesseq_def [simp]: x ≤ y ≡ ⟨x,y⟩ ∈  ROrd

  fixes sless (infix < 68)
  defines sless_def [simp]: x < y ≡ x≤y ∧ x≠y

  fixes nonnegative (ℝ⁺)
  defines nonnegative_def[simp]: ℝ⁺ ≡ Nonnegative(ℝ,Add, ROrd)

  fixes positiveset (ℝ₊)
  defines positiveset_def[simp]: ℝ₊ ≡ PositiveSet(ℝ,Add, ROrd)

  fixes setinv (- _ 72)
  defines setninv_def [simp]: -A ≡ GroupInv(ℝ,Add)(A)

  fixes non_zero (ℝ₀)
  defines non_zero_def[simp]: ℝ₀ ≡ ℝ-{0}
```

```
  fixes abs (| _ |)
  defines abs_def [simp]: |x| ≡ AbsoluteValue(ℝ,Add,ROrd)(x)

  fixes dist
  defines dist_def[simp]: dist ≡ {⟨p,|fst(p) - snd(p)|⟩ . p ∈ ℝ×ℝ}

  fixes two (2)
  defines two_def[simp]: 2 ≡ 1 + 1

  fixes inv (_⁻¹  [96] 97)
  defines inv_def[simp]:
    x⁻¹ ≡ GroupInv(ℝ0,restrict(Mul,ℝ0×ℝ0))(x)

  fixes listsum (∑ _ 70)
  defines listsum_def[simp]: ∑s ≡ Fold(Add,0,s)

  fixes nat_mult (infix · 95)
  defines nat_mult_def [simp]: n·x ≡ ∑{⟨k,x⟩. k∈n}

  fixes realsq (_² [96] 97)
  defines realsq_def [simp]: x² ≡ x·x

  fixes oddext (_ °)
  defines oddext_def [simp]: f° ≡ OddExtension(ℝ,Add,ROrd,f)

  fixes disk
  defines disk_def [simp]: disk(c,r) ≡ Disk(ℝ,dist,ROrd,c,r)

  fixes rxn (ℕ)
  defines rxn_def [simp]: ℕ ≡ ⋂ {N ∈ Pow(ℝ). 1 ∈ N ∧ (∀n. n∈N ⟶ n+1
∈ N)}
```

The assumtions of the `field1` locale (that sets the context for ordered fields) hold in the `reals` locale

```
lemma (in reals) field1_is_valid: shows field1(ℝ, Add, Mul,ROrd)
proof
  from R_are_reals show IsAring(ℝ, Add, Mul) and Mul {is commutative
on} ℝ
    and ROrd ⊆ ℝ × ℝ and IsLinOrder(ℝ,  ROrd)
    and ∀x y. ∀z∈ℝ. ⟨x, y⟩ ∈ ROrd ⟶ ⟨Add⟨x, z⟩, Add⟨y, z⟩⟩ ∈ ROrd
    and Nonnegative(ℝ, Add, ROrd) {is closed under} Mul
    and TheNeutralElement(ℝ, Add) ≠ TheNeutralElement(ℝ, Mul)
    and ∀x∈ℝ. x ≠ TheNeutralElement(ℝ,Add) ⟶ (∃y∈ℝ. Mul⟨x, y⟩ = TheNeutralElement(ℝ,Mu
    using IsAmodelOfReals_def IsAnOrdField_def IsAnOrdRing_def by auto
qed
```

We can use theorems proven in the `field1` locale in the `reals` locale. Note that since the the `field1` locale is an extension of the `ring1` locale, which is an extension of `ring0` locale , this makes available also the theorems proven in the `ring1` and `ring0` locales.

```
sublocale reals < field1 Reals Add Mul realadd realminus realsub realmul

  zero one two realsq listsum nat_mult ROrd
  using field1_is_valid by auto
```

The `group3` locale from the `OrderedGroup_ZF` theory defines context for theorems about ordered groups. We can use theorems proven in there in the `reals` locale as real numbers with addition form an ordered group.

```
sublocale reals < group3 Reals Add ROrd zero realadd realminus lesseq
sless nonnegative positiveset
  unfolding group3_def using OrdRing_ZF_1_L4 by auto
```

Since real numbers with addition form a group we can use the theorems proven in the `group0` locale defined in the `Group_ZF` theory in the `reals` locale.

```
sublocale reals < group0 Reals Add zero realadd realminus listsum nat_mult
  unfolding group3_def using OrderedGroup_ZF_1_L1 by auto
```

Let's recall basic properties of the real line.

```
lemma (in reals) basic_props: shows  ROrd {is total on} ℝ and Add {is
commutative on} ℝ
  using OrdRing_ZF_1_L4(2,3) by auto
```

The distance function `dist` defined in the `reals` locale is a metric.

```
lemma (in reals) dist_is_metric: shows
  dist : ℝ×ℝ → ℝ⁺
  ∀x∈ℝ.∀y∈ℝ. dist⟨x,y⟩ = |x - y|
  ∀x∈ℝ.dist⟨x,x⟩ = 0
  ∀x∈ℝ.∀y∈ℝ. dist⟨x,y⟩ = dist⟨y,x⟩
  ∀x∈ℝ.∀y∈ℝ.∀z∈ℝ. |x - z| ≤ |x - y| + |y - z|
  ∀x∈ℝ.∀y∈ℝ.∀z∈ℝ. dist⟨x,z⟩ ≤ dist⟨x, y⟩ + dist⟨y,z⟩
  ∀x∈ℝ.∀y∈ℝ. dist⟨x,y⟩ = 0 ⟶ x=y
  IsApseudoMetric(dist,ℝ,ℝ,Add,ROrd)
  IsAmetric(dist,ℝ,ℝ,Add,ROrd)
proof -
  show I: dist : ℝ×ℝ → ℝ⁺ using add_group.group_op_closed add_group.inverse_in_group
OrdRing_ZF_1_L4
      OrderedGroup_ZF_3_L3B  ZF_fun_from_total by simp
  then show II:∀x∈ℝ.∀y∈ℝ. dist⟨x,y⟩ = |x-y| using ZF_fun_from_tot_val0
by auto
  then show III: ∀x∈ℝ.dist⟨x,x⟩ = 0 using add_group.group0_2_L6 OrderedGroup_ZF_3_L2A
by simp
  { fix x y
    assume x∈ℝ y∈ℝ
    then have (-(x-y)) = y-x using add_group.group0_2_L12 by simp
```

```
    moreover from ‹x∈ℝ› ‹y∈ℝ› have |-(x-y)| =|x-y|
      using add_group.group_op_closed add_group.inverse_in_group basic_props(1)
OrderedGroup_ZF_3_L7A
      by simp
    ultimately have |y-x| = |x-y| by simp
    with ‹x∈ℝ› ‹y∈ℝ› II have dist⟨x,y⟩ = dist⟨y,x⟩ by simp
  } thus IV: ∀x∈ℝ.∀y∈ℝ. dist⟨x,y⟩ = dist⟨y,x⟩ by simp
  { fix x y
    assume x∈ℝ y∈ℝ dist⟨x,y⟩ = 0
    with II have |x-y| = 0 by simp
    with ‹x∈ℝ› ‹y∈ℝ› have x-y = 0
      using add_group.group_op_closed add_group.inverse_in_group OrderedGroup_ZF_3_L3D
by auto
    with ‹x∈ℝ› ‹y∈ℝ› havex=y using add_group.group0_2_L11A by simp
  } thus V: ∀x∈ℝ.∀y∈ℝ. dist⟨x,y⟩ = 0 ⟶ x=y by auto
  { fix x y z
    assume x∈ℝ y∈ℝ z∈ℝ
    then have |x-z| = |(x-y)+(y - z)| using add_group.cancel_middle(5)
by simp
    with ‹x∈ℝ› ‹y∈ℝ› ‹z∈ℝ› have |x-z|  ≤ |x-y| + |y - z|
      using add_group.group_op_closed add_group.inverse_in_group OrdRing_ZF_1_L4(2,3)
OrdGroup_triangle_ineq
      by simp
  } thus  ∀x∈ℝ.∀y∈ℝ.∀z∈ℝ. |x - z| ≤ |x - y| + |y - z| by simp
  with II show ∀x∈ℝ.∀y∈ℝ.∀z∈ℝ. dist⟨x,z⟩ ≤ dist⟨x, y⟩ + dist⟨y,z⟩ by
auto
  with I III IV V show IsApseudoMetric(dist,ℝ,ℝ,Add,ROrd) and IsAmetric(dist,ℝ,ℝ,Add,ROr
    unfolding IsApseudoMetric_def IsAmetric_def by auto
qed
```

Real numbers form an ordered loop.

```
lemma (in reals) reals_loop: shows IsAnOrdLoop(ℝ,Add,ROrd)
proof -
  have IsAloop(ℝ,Add) using add_group.group_is_loop by simp
  moreover from R_are_reals have ROrd ⊆ ℝ × ℝ and IsPartOrder(ℝ,ROrd)
    using IsAmodelOfReals_def IsAnOrdField_def IsAnOrdRing_def Order_ZF_1_L2

    by auto
  moreover
  { fix x y z assume A: x∈ℝ y∈ℝ  z∈ℝ
    then have x≤y ⟷ x+z ≤ y+z
      using ord_transl_inv ineq_cancel_right by blast
    moreover from A have x≤y ⟷ z+x ≤ z+y
      using ord_transl_inv OrderedGroup_ZF_1_L5AE by blast
    ultimately have (x≤y ⟷ x+z ≤ y+z) ∧ (x≤y ⟷ z+x ≤ z+y)
      by simp
  }
  ultimately show IsAnOrdLoop(ℝ,Add,ROrd) unfolding IsAnOrdLoop_def by
auto
```

qed

The assumptions of the `pmetric_space` locale hold in the `reals` locale.

```
lemma (in reals) pmetric_space_valid: shows pmetric_space(ℝ,Add, ROrd,dist,ℝ)

  unfolding pmetric_space_def pmetric_space_axioms_def loop1_def
  using reals_loop dist_is_metric(8)
  by blast
```

The assumptions of the `metric_space` locale hold in the `reals` locale.

```
lemma (in reals) metric_space_valid: shows metric_space(ℝ,Add, ROrd,dist,ℝ)
proof -
  have ∀x∈ℝ. ∀y∈ℝ. dist⟨x,y⟩=0 ⟶ x=y
    using dist_is_metric(9) unfolding IsAmetric_def by auto
  then show thesis unfolding metric_space_def metric_space_axioms_def

    using pmetric_space_valid by simp
qed
```

Some properties of the order relation on reals:

```
lemma (in reals) pos_is_lattice: shows
  IsLinOrder(ℝ,ROrd)
  IsLinOrder(ℝ₊,ROrd ∩ ℝ₊×ℝ₊)
  (ROrd ∩ ℝ₊×ℝ₊) {is a lattice on} ℝ₊
proof -
  show IsLinOrder(ℝ,ROrd) using OrdRing_ZF_1_L1 unfolding IsAnOrdRing_def
by simp
  moreover have ℝ₊ ⊆ ℝ using pos_set_in_gr by simp
  ultimately show IsLinOrder(ℝ₊,ROrd ∩ ℝ₊×ℝ₊) using ord_linear_subset(2)
by simp
  moreover have (ROrd ∩ ℝ₊×ℝ₊) ⊆ ℝ₊×ℝ₊ by auto
  ultimately show (ROrd ∩ ℝ₊×ℝ₊) {is a lattice on} ℝ₊ using lin_is_latt
by simp
qed
```

Of course the set of positive real numbers is nonempty as one is there.

```
lemma (in reals) pos_non_empty: shows ℝ₊≠0
  using R_are_reals ordring_one_is_pos
  unfolding IsAmodelOfReals_def IsAnOrdField_def by auto
```

We say that a relation r `down-directs` a set R if every two-element subset of R has a lower bound. The next lemma states that the natural order relation on real numbers down-directs the set of positive reals.

```
lemma (in reals) rord_down_directs: shows ROrd {down-directs} ℝ₊
  using pos_is_lattice(3) pos_non_empty meet_down_directs down_dir_mono
  unfolding IsAlattice_def by blast
```

We define the topology on reals as the metric topology coming from the `dist` metric (i.e. consisting of the unions of open disks).

```
definition (in reals) RealTopology (τℝ)
  where τℝ ≡ MetricTopology(ℝ,ℝ,Add,ROrd,dist)
```

A more explicit definition of the real topology in notation used in the `reals` context.

```
lemma (in reals) real_toplology_def_alt:
  shows τℝ = {⋃A. A ∈ Pow(⋃c∈ℝ.{disk(c,r). r ∈ ℝ₊})}
  unfolding MetricTopology_def RealTopology_def by simp
```

Real numbers form a Hausdorff topological space with topology generated by open disks.

```
theorem (in reals) reals_is_top:
  shows τℝ {is a topology} ⋃τℝ = ℝ τℝ {is T₂}
  using rord_down_directs metric_space_valid pmetric_space_valid pmetric_space.pmetric_is_to

    pmetric_space.metric_top_carrier metric_space.metric_space_T2
  unfolding RealTopology_def by simp_all

end
```

72 Complex numbers

```
theory Complex_ZF imports func_ZF_1 OrderedField_ZF

begin
```

The goal of this theory is to define complex numbers and prove that the Metamath complex numbers axioms hold.

72.1 From complete ordered fields to complex numbers

This section consists mostly of definitions and a proof context for talking about complex numbers. Suppose we have a set R with binary operations A and M and a relation r such that the quadruple (R, A, M, r) forms a complete ordered field. The next definitions take (R, A, M, r) and construct the sets that represent the structure of complex numbers: the carrier ($\mathbb{C} = R \times R$), binary operations of addition and multiplication of complex numbers and the order relation on $\mathbb{R} = R \times 0$. The `ImCxAdd, ReCxAdd, ImCxMul, ReCxMul` are helper meta-functions representing the imaginary part of a sum of complex numbers, the real part of a sum of real numbers, the imaginary part of a product of complex numbers and the real part of a product of real numbers, respectively. The actual operations (subsets of $(R \times R) \times R$ are named `CplxAdd` and `CplxMul`.

When R is an ordered field, it comes with an order relation. This induces a natural strict order relation on $\{\langle x, 0\rangle : x \in R\} \subseteq R \times R$. We call the set $\{\langle x,0\rangle : x \in R\}$ `ComplexReals(R,A)` and the strict order relation `CplxROrder(R,A,r)`. The order on the real axis of complex numbers is defined as the relation induced on it by the canonical projection on the first coordinate and the order we have on the real numbers. OK, lets repeat this slower. We start with the order relation r on a (model of) real numbers R. We want to define an order relation on a subset of complex numbers, namely on $R\times\{0\}$. To do that we use the notion of a relation induced by a mapping. The mapping here is $f : R \times \{0\} \to R, f\langle x,0\rangle = x$ which is defined under a name of `SliceProjection` in `func_ZF.thy`. This defines a relation r_1 (called `InducedRelation(f,r)`, see `func_ZF`) on $R \times \{0\}$ such that $\langle\langle x,0\rangle,\langle y,0\rangle \in r_1$ iff $\langle x,y\rangle \in r$. This way we get what we call `CplxROrder(R,A,r)`. However, this is not the end of the story, because Metamath uses strict inequalities in its axioms, rather than weak ones like IsarMathLib (mostly). So we need to take the strict version of this order relation. This is done in the syntax definition of $<_{\mathbb{R}}$ in the definition of `complex0` context. Since Metamath proves a lot of theorems about the real numbers extended with $+\infty$ and $-\infty$, we define the notation for inequalites on the extended real line as well.

A helper expression representing the real part of the sum of two complex numbers.

definition
```
ReCxAdd(R,A,a,b) ≡ A⟨fst(a),fst(b)⟩
```

An expression representing the imaginary part of the sum of two complex numbers.

definition
```
ImCxAdd(R,A,a,b) ≡ A⟨snd(a),snd(b)⟩
```

The set (function) that is the binary operation that adds complex numbers.

definition
```
CplxAdd(R,A) ≡
{⟨p, ⟨ ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p)) ⟩ ⟩.
p∈(R×R)×(R×R)}
```

The expression representing the imaginary part of the product of complex numbers.

definition
```
ImCxMul(R,A,M,a,b) ≡ A⟨M⟨fst(a),snd(b)⟩, M⟨snd(a),fst(b)⟩ ⟩
```

The expression representing the real part of the product of complex numbers.

definition
```
ReCxMul(R,A,M,a,b) ≡
A⟨M⟨fst(a),fst(b)⟩,GroupInv(R,A)(M⟨snd(a),snd(b)⟩)⟩
```

The function (set) that represents the binary operation of multiplication of complex numbers.

definition

```
CplxMul(R,A,M) ≡
{ ⟨p, ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩ ⟩.

p ∈ (R×R)×(R×R)}
```

The definition real numbers embedded in the complex plane.

definition

```
ComplexReals(R,A) ≡ R×{TheNeutralElement(R,A)}
```

Definition of order relation on the real line.

definition

```
CplxROrder(R,A,r) ≡
InducedRelation(SliceProjection(ComplexReals(R,A)),r)
```

The next locale defines proof context and notation that will be used for complex numbers.

locale complex0 =
  fixes R **and** A **and** M **and** r
  assumes R_are_reals: IsAmodelOfReals(R,A,M,r)

  fixes complex (ℂ)
  defines complex_def[simp]: ℂ ≡ R×R

  fixes rone ($\mathbf{1}_R$)
  defines rone_def[simp]: $\mathbf{1}_R$ ≡ TheNeutralElement(R,M)

  fixes rzero ($\mathbf{0}_R$)
  defines rzero_def[simp]: $\mathbf{0}_R$ ≡ TheNeutralElement(R,A)

  fixes one (**1**)
  defines one_def[simp]: **1** ≡ ⟨$\mathbf{1}_R$, $\mathbf{0}_R$⟩

  fixes zero (**0**)
  defines zero_def[simp]: **0** ≡ ⟨$\mathbf{0}_R$, $\mathbf{0}_R$⟩

  fixes iunit (i)
  defines iunit_def[simp]: i ≡ ⟨$\mathbf{0}_R$,$\mathbf{1}_R$⟩

  fixes creal (ℝ)
  defines creal_def[simp]: ℝ ≡ {⟨r,$\mathbf{0}_R$⟩. r∈R}

  fixes rmul (**infixl** · 71)
  defines rmul_def[simp]: a · b ≡ M⟨a,b⟩

  fixes radd (**infixl** + 69)
  defines radd_def[simp]: a + b ≡ A⟨a,b⟩

  fixes rneg (- _ 70)

```
defines rneg_def[simp]: - a ≡  GroupInv(R,A)(a)

fixes ca (infixl + 69)
defines ca_def[simp]: a + b ≡ CplxAdd(R,A)⟨a,b⟩

fixes cm (infixl · 71)
defines cm_def[simp]: a · b ≡ CplxMul(R,A,M)⟨a,b⟩

fixes cdiv (infixl / 70)
defines cdiv_def[simp]: a / b ≡ ⋃ { x ∈ ℂ. b · x = a }

fixes sub (infixl - 69)
defines sub_def[simp]: a - b ≡ ⋃ { x ∈ ℂ. b + x = a }

fixes cneg (-_ 95)
defines cneg_def[simp]: - a ≡ 0 - a

fixes lessr (infix <ℝ 68)
defines lessr_def[simp]:
a <ℝ b ≡ ⟨a,b⟩ ∈ StrictVersion(CplxROrder(R,A,r))

fixes cpnf (+∞)
defines cpnf_def[simp]: +∞ ≡ ℂ

fixes cmnf (−∞)
defines cmnf_def[simp]: −∞ ≡ {ℂ}

fixes cxr (ℝ*)
defines cxr_def[simp]: ℝ* ≡ ℝ ∪ {+∞,−∞}

fixes cxn (ℕ)
defines cxn_def[simp]:
ℕ ≡ ⋂ {N ∈ Pow(ℝ). 1 ∈ N ∧ (∀n. n∈N ⟶ n+1 ∈ N)}

fixes cltrrset (<)
defines cltrrset_def[simp]:
< ≡  StrictVersion(CplxROrder(R,A,r)) ∩ ℝ×ℝ ∪
{⟨−∞,+∞⟩} ∪ (ℝ×{+∞}) ∪ ({−∞}×ℝ )

fixes cltrr (infix < 68)
defines cltrr_def[simp]: a < b ≡ ⟨a,b⟩ ∈ <

fixes lsq (infix ≤ 68)
defines lsq_def[simp]: a ≤ b ≡ ¬ (b < a)

fixes two (2)
defines two_def[simp]: 2 ≡ 1 + 1

fixes three (3)
```

```
  defines three_def[simp]: 3 ≡ 2+1

  fixes four (4)
  defines four_def[simp]: 4 ≡ 3+1

  fixes five (5)
  defines five_def[simp]: 5 ≡ 4+1

  fixes six (6)
  defines six_def[simp]: 6 ≡ 5+1

  fixes seven (7)
  defines seven_def[simp]: 7 ≡ 6+1

  fixes eight (8)
  defines eight_def[simp]: 8 ≡ 7+1

  fixes nine (9)
  defines nine_def[simp]: 9 ≡ 8+1
```

72.2 Axioms of complex numbers

In this section we will prove that all Metamath's axioms of complex numbers hold in the `complex0` context.

The next lemma lists some contexts that are valid in the `complex0` context.

```
lemma (in complex0) valid_cntxts: shows
  field1(R,A,M,r)
  field0(R,A,M)
  ring1(R,A,M,r)
  group3(R,A,r)
  ring0(R,A,M)
  M {is commutative on} R
  group0(R,A)
proof -
  from R_are_reals have I: IsAnOrdField(R,A,M,r)
    using IsAmodelOfReals_def by simp
  then show field1(R,A,M,r) using OrdField_ZF_1_L2 by simp
  then show ring1(R,A,M,r) and I: field0(R,A,M)
    using field1.axioms ring1_def field1.OrdField_ZF_1_L1B
    by auto
  then show group3(R,A,r) using ring1.OrdRing_ZF_1_L4
    by simp
  from I have IsAfield(R,A,M) using field0.Field_ZF_1_L1
    by simp
  then have IsAring(R,A,M) and M {is commutative on} R
    using IsAfield_def by auto
  then show ring0(R,A,M) and M {is commutative on} R
    using ring0_def by auto
```

```
  then show group0(R,A) using ring0.Ring_ZF_1_L1
    by simp
qed
```

The next lemma shows the definition of real and imaginary part of complex sum and product in a more readable form using notation defined in `complex0` locale.

```
lemma (in complex0) cplx_mul_add_defs: shows
  ReCxAdd(R,A,⟨a,b⟩,⟨c,d⟩) = a + c
  ImCxAdd(R,A,⟨a,b⟩,⟨c,d⟩) = b + d
  ImCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) = a·d + b·c
  ReCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) =  a·c + (-b·d)
proof -
  let z₁ = ⟨a,b⟩
  let z₂ = ⟨c,d⟩
  have ReCxAdd(R,A,z₁,z₂) ≡  A⟨fst(z₁),fst(z₂)⟩
   by (rule ReCxAdd_def)
  moreover have ImCxAdd(R,A,z₁,z₂) ≡  A⟨snd(z₁),snd(z₂)⟩
    by (rule ImCxAdd_def)
  moreover have
    ImCxMul(R,A,M,z₁,z₂) ≡ A⟨M<fst(z₁),snd(z₂)>,M<snd(z₁),fst(z₂)>⟩
    by (rule ImCxMul_def)
  moreover have
    ReCxMul(R,A,M,z₁,z₂) ≡
    A⟨M<fst(z₁),fst(z₂)>,GroupInv(R,A)(M⟨snd(z₁),snd(z₂)⟩)⟩
    by (rule ReCxMul_def)
  ultimately show
    ReCxAdd(R,A,z₁,z₂) =  a + c
    ImCxAdd(R,A,⟨a,b⟩,⟨c,d⟩) = b + d
    ImCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) = a·d + b·c
    ReCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) =  a·c + (-b·d)
    by auto
qed
```

Real and imaginary parts of sums and products of complex numbers are real.

```
lemma (in complex0) cplx_mul_add_types:
  assumes A1: z₁ ∈ ℂ   z₂ ∈ ℂ
  shows
  ReCxAdd(R,A,z₁,z₂) ∈ R
  ImCxAdd(R,A,z₁,z₂) ∈ R
  ImCxMul(R,A,M,z₁,z₂) ∈ R
  ReCxMul(R,A,M,z₁,z₂) ∈ R
proof -
  let a = fst(z₁)
  let b = snd(z₁)
  let c = fst(z₂)
  let d = snd(z₂)
  from A1 have a ∈ R  b ∈ R  c ∈ R  d ∈ R
```

```
    by auto
  then have
    a + c ∈ R
    b + d ∈ R
    a·d + b·c ∈ R
    a·c + (- b·d) ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 by auto
  with A1 show
    ReCxAdd(R,A,z₁,z₂) ∈ R
    ImCxAdd(R,A,z₁,z₂) ∈ R
    ImCxMul(R,A,M,z₁,z₂) ∈ R
    ReCxMul(R,A,M,z₁,z₂) ∈ R
    using cplx_mul_add_defs by auto
qed
```

Complex reals are complex. Recall the definition of ℝ in the complex0 locale.

```
lemma (in complex0) axresscn: shows ℝ ⊆ ℂ
  using valid_cntxts group0.group0_2_L2 by auto
```

Complex 1 is not complex 0.

```
lemma (in complex0) ax1ne0: shows 1 ≠ 0
proof -
  have IsAfield(R,A,M) using valid_cntxts field0.Field_ZF_1_L1
    by simp
  then show 1 ≠ 0 using IsAfield_def by auto
qed
```

Complex addition is a complex valued binary operation on complex numbers.

```
lemma (in complex0) axaddopr: shows CplxAdd(R,A): ℂ × ℂ → ℂ
proof -
  have ∀p ∈ ℂ×ℂ.
    ⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩ ∈ ℂ
    using cplx_mul_add_types by simp
  then have
    {⟨p,⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩ ⟩.
    p ∈ ℂ×ℂ}: ℂ×ℂ → ℂ
    by (rule ZF_fun_from_total)
  then show CplxAdd(R,A): ℂ × ℂ → ℂ using CplxAdd_def by simp
qed
```

Complex multiplication is a complex valued binary operation on complex numbers.

```
lemma (in complex0) axmulopr: shows CplxMul(R,A,M): ℂ × ℂ → ℂ
proof -
  have ∀p ∈ ℂ×ℂ.
    ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩ ∈ ℂ
    using cplx_mul_add_types by simp
  then have
```

```
  {⟨p,⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩⟩.
    p ∈ ℂ×ℂ}: ℂ×ℂ → ℂ by (rule ZF_fun_from_total)
  then show CplxMul(R,A,M): ℂ × ℂ → ℂ using CplxMul_def by simp
qed
```

What are the values of omplex addition and multiplication in terms of their real and imaginary parts?

```
lemma (in complex0) cplx_mul_add_vals:
  assumes A1: a∈R  b∈R  c∈R  d∈R
  shows
  ⟨a,b⟩ + ⟨c,d⟩ = ⟨a + c, b + d⟩
  ⟨a,b⟩ · ⟨c,d⟩ = ⟨a·c + (-b·d), a·d + b·c⟩
proof -
  let S = CplxAdd(R,A)
  let P = CplxMul(R,A,M)
  let p = ⟨ ⟨a,b⟩, ⟨c,d⟩ ⟩
  from A1 have S : ℂ × ℂ → ℂ and p ∈ ℂ × ℂ
    using axaddopr by auto
  moreover have
    S = {⟨p, <ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))>⟩.

    p ∈ ℂ × ℂ}
    using CplxAdd_def by simp
  ultimately have S(p) = ⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩
    by (rule ZF_fun_from_tot_val)
  then show ⟨a,b⟩ + ⟨c,d⟩ = ⟨a + c, b + d⟩
    using cplx_mul_add_defs by simp
  from A1 have P : ℂ × ℂ → ℂ and p ∈ ℂ × ℂ
    using axmulopr by auto
  moreover have
    P = {⟨p, ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩
⟩.
    p ∈ ℂ × ℂ}
    using CplxMul_def by simp
  ultimately have
    P(p) = ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩
    by (rule ZF_fun_from_tot_val)
  then show ⟨a,b⟩ · ⟨c,d⟩ = ⟨a·c + (-b·d), a·d + b·c⟩
    using cplx_mul_add_defs by simp
qed
```

Complex multiplication is commutative.

```
lemma (in complex0) axmulcom: assumes A1: a ∈ ℂ  b ∈ ℂ
  shows a·b = b·a
  using assms cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L4
    field0.field_mult_comm by auto
```

A sum of complex numbers is complex.

```
lemma (in complex0) axaddcl: assumes a ∈ ℂ  b ∈ ℂ
```

```
  shows a+b ∈ ℂ
  using assms axaddopr apply_funtype by simp
```

A product of complex numbers is complex.

```
lemma (in complex0) axmulcl: assumes a ∈ ℂ  b ∈ ℂ
  shows  a·b ∈ ℂ
  using assms axmulopr apply_funtype by simp
```

Multiplication is distributive with respect to addition.

```
lemma (in complex0) axdistr:
  assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a·(b + c) = a·b + a·c
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let b_r = fst(b)
  let b_i = snd(b)
  let c_r = fst(c)
  let c_i = snd(c)
  from A1 have T:
    a_r ∈ R  a_i ∈ R  b_r ∈ R  b_i ∈ R  c_r ∈ R  c_i ∈ R
    b_r+c_r ∈ R  b_i+c_i ∈ R
    a_r·b_r + (-a_i·b_i) ∈ R
    a_r·c_r + (-a_i·c_i) ∈ R
    a_r·b_i + a_i·b_r ∈ R
    a_r·c_i + a_i·c_r ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 by auto
  with A1 have a·(b + c) =
    ⟨a_r·(b_r+c_r) + (-a_i·(b_i+c_i)),a_r·(b_i+c_i) + a_i·(b_r+c_r)⟩
    using cplx_mul_add_vals by auto
  moreover from T have
    a_r·(b_r+c_r) + (-a_i·(b_i+c_i)) =
    a_r·b_r + (-a_i·b_i) + (a_r·c_r + (-a_i·c_i))
    and
    a_r·(b_i+c_i) + a_i·(b_r+c_r) =
    a_r·b_i + a_i·b_r + (a_r·c_i + a_i·c_r)
    using valid_cntxts ring0.Ring_ZF_2_L6 by auto
  moreover from A1 T have
    ⟨a_r·b_r + (-a_i·b_i) + (a_r·c_r + (-a_i·c_i)),
    a_r·b_i + a_i·b_r + (a_r·c_i + a_i·c_r)⟩ =
    a·b + a·c
    using cplx_mul_add_vals by auto
  ultimately show a·(b + c) = a·b + a·c
    by simp
qed
```

Complex addition is commutative.

```
lemma (in complex0) axaddcom: assumes a ∈ ℂ  b ∈ ℂ
  shows a+b = b+a
```

```
  using assms cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L4
  by auto
```

Complex addition is associative.

```
lemma (in complex0) axaddass: assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a + b + c = a + (b + c)
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let b_r = fst(b)
  let b_i = snd(b)
  let c_r = fst(c)
  let c_i = snd(c)
  from A1 have T:
    a_r ∈ R  a_i ∈ R  b_r ∈ R  b_i ∈ R  c_r ∈ R  c_i ∈ R
    a_r+b_r ∈ R  a_i+b_i ∈ R
    b_r+c_r ∈ R  b_i+c_i ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 by auto
  with A1 have a + b + c = ⟨a_r+b_r+c_r,a_i+b_i+c_i⟩
    using cplx_mul_add_vals by auto
  also from A1 T have ... = a + (b + c)
    using valid_cntxts ring0.Ring_ZF_1_L11 cplx_mul_add_vals
    by auto
  finally show a + b + c = a + (b + c)
    by simp
qed
```

Complex multiplication is associative.

```
lemma (in complex0) axmulass: assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a · b · c = a · (b · c)
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let b_r = fst(b)
  let b_i = snd(b)
  let c_r = fst(c)
  let c_i = snd(c)
  from A1 have T:
    a_r ∈ R  a_i ∈ R  b_r ∈ R  b_i ∈ R  c_r ∈ R  c_i ∈ R
    a_r·b_r + (-a_i·b_i) ∈ R
    a_r·b_i + a_i·b_r ∈ R
    b_r·c_r + (-b_i·c_i) ∈ R
    b_r·c_i + b_i·c_r ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4  by auto
  with A1 have a · b · c =
    ⟨(a_r·b_r + (-a_i·b_i))·c_r + (-(a_r·b_i + a_i·b_r)·c_i),
    (a_r·b_r + (-a_i·b_i))·c_i + (a_r·b_i + a_i·b_r)·c_r⟩
    using cplx_mul_add_vals by auto
  moreover from A1 T have
```

```
    ⟨a_r·(b_r·c_r + (-b_i·c_i)) + (-a_i·(b_r·c_i + b_i·c_r)),
    a_r·(b_r·c_i + b_i·c_r) + a_i·(b_r·c_r + (-b_i·c_i))⟩ =
    a · (b · c)
    using cplx_mul_add_vals by auto
  moreover from T have
    a_r·(b_r·c_r + (-b_i·c_i)) + (-a_i·(b_r·c_i + b_i·c_r)) =
    (a_r·b_r + (-a_i·b_i))·c_r + (-(a_r·b_i + a_i·b_r)·c_i)
    and
    a_r·(b_r·c_i + b_i·c_r) + a_i·(b_r·c_r + (-b_i·c_i)) =
    (a_r·b_r + (-a_i·b_i))·c_i + (a_r·b_i + a_i·b_r)·c_r
    using valid_cntxts ring0.Ring_ZF_2_L6 by auto
  ultimately show a · b · c = a · (b · c)
    by auto
qed
```

Complex 1 is real. This really means that the pair $\langle 1, 0\rangle$ is on the real axis.

```
lemma (in complex0) ax1re: shows 1 ∈ ℝ
  using valid_cntxts ring0.Ring_ZF_1_L2 by simp
```

The imaginary unit is a "square root" of -1 (that is, $i^2 + 1 = 0$).

```
lemma (in complex0) axi2m1: shows i·i + 1 = 0
  using valid_cntxts ring0.Ring_ZF_1_L2 ring0.Ring_ZF_1_L3
  cplx_mul_add_vals ring0.Ring_ZF_1_L6 group0.group0_2_L6
  by simp
```

0 is the neutral element of complex addition.

```
lemma (in complex0) ax0id: assumes a ∈ ℂ
  shows a + 0 = a
  using assms cplx_mul_add_vals valid_cntxts
    ring0.Ring_ZF_1_L2 ring0.Ring_ZF_1_L3
  by auto
```

The imaginary unit is a complex number.

```
lemma (in complex0) axicn: shows i ∈ ℂ
  using valid_cntxts ring0.Ring_ZF_1_L2 by auto
```

All complex numbers have additive inverses.

```
lemma (in complex0) axnegex: assumes A1: a ∈ ℂ
  shows ∃x∈ℂ. a + x  = 0
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let x = ⟨-a_r, -a_i⟩
  from A1 have T:
    a_r ∈ R   a_i ∈ R   (-a_r) ∈ R   (-a_r) ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L3 by auto
  then have x ∈ ℂ using valid_cntxts ring0.Ring_ZF_1_L3
    by auto
```

```
  moreover from A1 T have a + x = 0
    using cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L3
    by auto
  ultimately show ∃x∈ℂ. a + x  = 0
    by auto
qed
```

A non-zero complex number has a multiplicative inverse.

```
lemma (in complex0) axrecex: assumes A1: a ∈ ℂ and A2: a≠0
  shows ∃x∈ℂ. a·x = 1
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let m = a_r·a_r + a_i·a_i
  from A1 have T1: a_r ∈ R   a_i ∈ R by auto
  moreover from A1 A2 have a_r ≠ 0_R ∨ a_i ≠ 0_R
    by auto
  ultimately have ∃c∈R. m·c = 1_R
    using valid_cntxts field1.OrdField_ZF_1_L10
    by auto
  then obtain c where I: c∈R and II: m·c = 1_R
    by auto
  let x = ⟨a_r·c, -a_i·c⟩
  from T1 I have T2: a_r·c ∈ R  (-a_i·c) ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 ring0.Ring_ZF_1_L3
    by auto
  then have x ∈ ℂ by auto
  moreover from A1 T1 T2 I II have a·x = 1
    using cplx_mul_add_vals valid_cntxts ring0.ring_rearr_3_elemA
    by auto
  ultimately show ∃x∈ℂ. a·x = 1 by auto
qed
```

Complex 1 is a right neutral element for multiplication.

```
lemma (in complex0) ax1id: assumes A1: a ∈ ℂ
  shows a·1 = a
  using assms valid_cntxts ring0.Ring_ZF_1_L2 cplx_mul_add_vals
    ring0.Ring_ZF_1_L3 ring0.Ring_ZF_1_L6 by auto
```

A formula for sum of (complex) real numbers.

```
lemma (in complex0) sum_of_reals: assumes a∈ℝ  b∈ℝ
  shows
  a + b = ⟨fst(a) + fst(b),0_R⟩
  using assms valid_cntxts ring0.Ring_ZF_1_L2 cplx_mul_add_vals
    ring0.Ring_ZF_1_L3 by auto
```

The sum of real numbers is real.

```
lemma (in complex0) axaddrcl: assumes A1: a∈ℝ  b∈ℝ
```

```
  shows a + b ∈ ℝ
  using assms sum_of_reals valid_cntxts ring0.Ring_ZF_1_L4
  by auto
```

The formula for the product of (complex) real numbers.

```
lemma (in complex0) prod_of_reals: assumes A1: a∈ℝ  b∈ℝ
  shows a · b = ⟨fst(a)·fst(b),0_R⟩
proof -
  let a_r = fst(a)
  let b_r = fst(b)
  from A1 have T:
    a_r ∈ R b_r ∈ R  0_R ∈ R  a_r·b_r ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L2 ring0.Ring_ZF_1_L4
    by auto
  with A1 show a · b = ⟨a_r·b_r,0_R⟩
    using cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L2
      ring0.Ring_ZF_1_L6 ring0.Ring_ZF_1_L3 by auto
qed
```

The product of (complex) real numbers is real.

```
lemma (in complex0) axmulrcl: assumes a∈ℝ  b∈ℝ
  shows a · b ∈ ℝ
  using assms prod_of_reals valid_cntxts ring0.Ring_ZF_1_L4
  by auto
```

The existence of a real negative of a real number.

```
lemma (in complex0) axrnegex: assumes A1: a∈ℝ
  shows ∃ x ∈ ℝ. a + x = 0
proof -
  let a_r = fst(a)
  let x = ⟨-a_r,0_R⟩
  from A1 have T:
    a_r ∈ R  (-a_r) ∈ R  0_R ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L3 ring0.Ring_ZF_1_L2
    by auto
  then have x∈ ℝ by auto
  moreover from A1 T have a + x = 0
    using cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L3
    by auto
  ultimately show ∃x∈ℝ. a + x = 0 by auto
qed
```

Each nonzero real number has a real inverse

```
lemma (in complex0) axrrecex:
  assumes A1:  a ∈ ℝ   a ≠ 0
  shows ∃x∈ℝ. a · x = 1
proof -
  let R_0 = R-{0_R}
```

let a_r = fst(a)
let y = GroupInv(R_0,restrict(M,$R_0 \times R_0$))(a_r)
from A1 **have** T: $\langle y,\mathbf{0}_R\rangle \in \mathbb{R}$ **using** valid_cntxts field0.Field_ZF_1_L5
by auto
moreover from A1 T **have** a $\cdot$ $\langle y,\mathbf{0}_R\rangle$ = **1**
using prod_of_reals valid_cntxts
field0.Field_ZF_1_L5 field0.Field_ZF_1_L6 **by** auto
ultimately show $\exists$ x $\in \mathbb{R}$. a $\cdot$ x = **1** **by** auto
qed

Our $\mathbb{R}$ symbol is the real axis on the complex plane.

lemma (in complex0) real_means_real_axis: **shows** $\mathbb{R}$ = ComplexReals(R,A)
using ComplexReals_def **by** auto

The CplxROrder thing is a relation on the complex reals.

lemma (in complex0) cplx_ord_on_cplx_reals:
shows CplxROrder(R,A,r) $\subseteq \mathbb{R}\times\mathbb{R}$
using ComplexReals_def slice_proj_bij real_means_real_axis
CplxROrder_def InducedRelation_def **by** auto

The strict version of the complex relation is a relation on complex reals.

lemma (in complex0) cplx_strict_ord_on_cplx_reals:
shows StrictVersion(CplxROrder(R,A,r)) $\subseteq \mathbb{R}\times\mathbb{R}$
using cplx_ord_on_cplx_reals strict_ver_rel **by** simp

The CplxROrder thing is a relation on the complex reals. Here this is formulated as a statement that in complex0 context $a < b$ implies that a, b are complex reals

lemma (in complex0) strict_cplx_ord_type: **assumes** a $<_{\mathbb{R}}$ b
shows a$\in\mathbb{R}$ b$\in\mathbb{R}$
using assms CplxROrder_def def_of_strict_ver InducedRelation_def
slice_proj_bij ComplexReals_def real_means_real_axis
by auto

A more readable version of the definition of the strict order relation on the real axis. Recall that in the complex0 context r denotes the (non-strict) order relation on the underlying model of real numbers.

lemma (in complex0) def_of_real_axis_order: **shows**
$\langle x,\mathbf{0}_R\rangle <_{\mathbb{R}} \langle y,\mathbf{0}_R\rangle \longleftrightarrow \langle x,y\rangle \in r \wedge x\neq y$
proof
let f = SliceProjection(ComplexReals(R,A))
assume A1: $\langle x,\mathbf{0}_R\rangle <_{\mathbb{R}} \langle y,\mathbf{0}_R\rangle$
then have $\langle$ f$\langle x,\mathbf{0}_R\rangle$, f$\langle y,\mathbf{0}_R\rangle$ $\rangle \in r \wedge x \neq y$
using CplxROrder_def def_of_strict_ver def_of_ind_relA
by simp
moreover from A1 **have** $\langle x,\mathbf{0}_R\rangle \in \mathbb{R}$ $\langle y,\mathbf{0}_R\rangle \in \mathbb{R}$
using strict_cplx_ord_type **by** auto
ultimately show $\langle x,y\rangle \in r \wedge x\neq y$

```
    using slice_proj_bij ComplexReals_def by simp
next assume A1: ⟨x,y⟩ ∈ r ∧ x≠y
  let f = SliceProjection(ComplexReals(R,A))
  have f : ℝ → R
    using ComplexReals_def slice_proj_bij real_means_real_axis
    by simp
  moreover from A1 have T: ⟨x,0_R⟩ ∈ ℝ   ⟨y,0_R⟩ ∈ ℝ
    using valid_cntxts ring1.OrdRing_ZF_1_L3 by auto
  moreover from A1 T have ⟨ f⟨x,0_R⟩, f⟨y,0_R⟩ ⟩ ∈ r
    using slice_proj_bij ComplexReals_def by simp
  ultimately have ⟨ ⟨x,0_R⟩, ⟨y,0_R⟩ ⟩ ∈ InducedRelation(f,r)
    using def_of_ind_relB by simp
  with A1 show ⟨x,0_R⟩ <_ℝ ⟨y,0_R⟩
    using CplxROrder_def def_of_strict_ver
    by simp
qed
```

The (non strict) order on complex reals is antisymmetric, transitive and total.

```
lemma (in complex0) cplx_ord_antsym_trans_tot: shows
  antisym(CplxROrder(R,A,r))
  trans(CplxROrder(R,A,r))
  CplxROrder(R,A,r) {is total on} ℝ
proof -
  let f = SliceProjection(ComplexReals(R,A))
  have f ∈ ord_iso(ℝ,CplxROrder(R,A,r),R,r)
    using ComplexReals_def slice_proj_bij real_means_real_axis
      bij_is_ord_iso CplxROrder_def by simp
  moreover have CplxROrder(R,A,r) ⊆ ℝ×ℝ
    using cplx_ord_on_cplx_reals by simp
  moreover have I:
    antisym(r)   r {is total on} R   trans(r)
    using valid_cntxts ring1.OrdRing_ZF_1_L1 IsAnOrdRing_def
      IsLinOrder_def by auto
  ultimately show
    antisym(CplxROrder(R,A,r))
    trans(CplxROrder(R,A,r))
    CplxROrder(R,A,r) {is total on} ℝ
    using ord_iso_pres_antsym ord_iso_pres_tot ord_iso_pres_trans
    by auto
qed
```

The trichotomy law for the strict order on the complex reals.

```
lemma (in complex0) cplx_strict_ord_trich:
  assumes a ∈ ℝ   b ∈ ℝ
  shows Exactly_1_of_3_holds(a<_ℝb, a=b, b<_ℝa)
  using assms cplx_ord_antsym_trans_tot strict_ans_tot_trich
  by simp
```

The strict order on the complex reals is kind of antisymetric.

```
lemma (in complex0) pre_axlttri: assumes A1: a ∈ ℝ   b ∈ ℝ
  shows a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a)
proof -
  from A1 have Exactly_1_of_3_holds(a<ℝb, a=b, b<ℝa)
    by (rule cplx_strict_ord_trich)
  then show a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a)
    by (rule Fol1_L8A)
qed
```

The strict order on complex reals is transitive.

```
lemma (in complex0) cplx_strict_ord_trans:
  shows trans(StrictVersion(CplxROrder(R,A,r)))
  using cplx_ord_antsym_trans_tot strict_of_transB by simp
```

The strict order on complex reals is transitive - the explicit version of `cplx_strict_ord_trans`.

```
lemma (in complex0) pre_axlttrn:
  assumes A1: a <ℝ b   b <ℝ c
  shows a <ℝ c
proof -
  let s = StrictVersion(CplxROrder(R,A,r))
  from A1 have
    trans(s)   ⟨a,b⟩ ∈ s ∧ ⟨b,c⟩ ∈ s
    using cplx_strict_ord_trans by auto
  then have ⟨a,c⟩ ∈ s by (rule Fol1_L3)
  then show a <ℝ c by simp
qed
```

The strict order on complex reals is preserved by translations.

```
lemma (in complex0) pre_axltadd:
  assumes A1: a <ℝ b and A2: c ∈ ℝ
  shows c+a <ℝ c+b
proof -
  from A1 have T: a∈ℝ  b∈ℝ using strict_cplx_ord_type
    by auto
  with A1 A2 show c+a <ℝ c+b
    using def_of_real_axis_order valid_cntxts
      group3.group_strict_ord_transl_inv sum_of_reals
    by auto
qed
```

The set of positive complex reals is closed with respect to multiplication.

```
lemma (in complex0) pre_axmulgt0: assumes A1: 0 <ℝ a   0 <ℝ b
  shows 0 <ℝ a·b
proof -
  from A1 have T: a∈ℝ  b∈ℝ using strict_cplx_ord_type
    by auto
```

```
  with A1 show 0 <ℝ a·b
    using def_of_real_axis_order valid_cntxts field1.pos_mul_closed
      def_of_real_axis_order prod_of_reals
    by auto
qed
```

The order on complex reals is linear and complete.

```
lemma (in complex0) cmplx_reals_ord_lin_compl: shows
  CplxROrder(R,A,r) {is complete}
  IsLinOrder(ℝ,CplxROrder(R,A,r))
proof -
  have SliceProjection(ℝ) ∈ bij(ℝ,R)
    using slice_proj_bij ComplexReals_def real_means_real_axis
    by simp
  moreover have r ⊆ R×R using valid_cntxts ring1.OrdRing_ZF_1_L1
    IsAnOrdRing_def by simp
  moreover from R_are_reals have
    r {is complete} and IsLinOrder(R,r)
    using IsAmodelOfReals_def valid_cntxts ring1.OrdRing_ZF_1_L1
    IsAnOrdRing_def by auto
  ultimately show
    CplxROrder(R,A,r) {is complete}
    IsLinOrder(ℝ,CplxROrder(R,A,r))
    using CplxROrder_def real_means_real_axis ind_rel_pres_compl
      ind_rel_pres_lin by auto
qed
```

The property of the strict order on complex reals that corresponds to completeness.

```
lemma (in complex0) pre_axsup: assumes A1: X ⊆ ℝ   X ≠ 0 and
  A2: ∃x∈ℝ. ∀y∈X. y <ℝ x
  shows
  ∃x∈ℝ. (∀y∈X. ¬(x <ℝ y)) ∧ (∀y∈ℝ. (y <ℝ x ⟶ (∃z∈X. y <ℝ z)))
proof -
  let s = StrictVersion(CplxROrder(R,A,r))
  have
    CplxROrder(R,A,r) ⊆ ℝ×ℝ
    IsLinOrder(ℝ,CplxROrder(R,A,r))
    CplxROrder(R,A,r) {is complete}
    using cplx_ord_on_cplx_reals cmplx_reals_ord_lin_compl
    by auto
  moreover note A1
  moreover have s = StrictVersion(CplxROrder(R,A,r))
    by simp
  moreover from A2 have ∃u∈ℝ. ∀y∈X. ⟨y,u⟩ ∈ s
    by simp
  ultimately have
    ∃x∈ℝ. ( ∀y∈X. ⟨x,y⟩ ∉ s ) ∧
    (∀y∈ℝ. ⟨y,x⟩ ∈ s ⟶ (∃z∈X. ⟨y,z⟩ ∈ s))
```

```
    by (rule strict_of_compl)
  then show (∃x∈ℝ. (∀y∈X. ¬(x <ℝ y)) ∧
    (∀y∈ℝ. (y <ℝ x ⟶ (∃z∈X. y <ℝ z))))
    by simp
qed

end
```

73 Rings - Zariski Topology

This file deals with the definition of the topology on the set of prime ideals

It defines the topology, computes the closed sets and the closure and interior operators

```
theory Ring_Zariski_ZF imports Ring_ZF_2 Topology_ZF

begin
```

The set where the topology is defined is in the spectrum of a ring; i.e. the set of all prime ideals.

```
definition (in ring0) Spec where
Spec ≡ {I∈ℐ. I◁pR}
```

The basic set that defines the topoogy is given by the D operator

```
definition (in ring0) openBasic (D) where
S⊆R ⟹ D(S) ≡ {I∈Spec. ¬(S⊆I)}
```

The D operator preserves subsets

```
lemma (in ring0) D_operator_preserve_subset:
  assumes S ⊆ T T⊆R
  shows D(S) ⊆ D(T)
proof
  from assms have S:S⊆R by auto
  fix x assume x∈D(S)
  then have x:x∈Spec ¬(S⊆x) using openBasic_def S by auto
  with assms(1) have x∈Spec ¬(T⊆x) by auto
  then show x:D(T) using openBasic_def assms(2) by auto
qed
```

The D operator values can be obtained by considering only ideals. This is useful as we have operations on ideals that we do not have on subsets.

```
lemma (in ring0) D_operator_only_ideals:
  assumes T⊆R
  shows D(T) = D(⟨T⟩I)
proof
  have T:T⊆⟨T⟩I ⟨T⟩I ⊆R using generated_ideal_contains_set assms
    generated_ideal_is_ideal[OF assms] ideal_dest_subset by auto
```

```
  with D_operator_preserve_subset show D(T) ⊆ D(⟨T⟩_I)
    by auto
  {
    fix t assume t∈D(⟨T⟩_I)
    with T(2) have t:t∈Spec ¬(⟨T⟩_I ⊆t) using openBasic_def by auto
    {
      assume as:T ⊆ t
      from t(1) have t◁R unfolding Spec_def primeIdeal_def by auto
      with as have ⟨T⟩_I ⊆t using generated_ideal_small by auto
      with t(2) have False by auto
    }
    then have ¬(T ⊆ t) by auto
    with t(1) have t∈D(T) using openBasic_def assms by auto
  }
  then show D(⟨T⟩_I) ⊆ D(T) by auto
qed
```

The intersection of to D-sets is the D-set on the product of ideals

```
lemma (in ring0) intersection_open_basic:
  assumes I◁R J◁R
  shows D(I)∩D(J) = D(I·_I J)
proof
  have S:I·_I J ⊆ R using product_in_intersection(2) ideal_dest_subset assms
by auto
  {
    fix K assume K:K∈D(I)∩D(J)
    then have K◁_pR ¬(I⊆K) ¬(J⊆K)
      using assms ideal_dest_subset openBasic_def
      unfolding Spec_def by auto
    then have ¬(I⊆K) ¬(J⊆K) ∀I∈ℐ. ∀J∈ℐ. I·_I J⊆K ⟶ I ⊆ K ∨ J ⊆ K
      unfolding primeIdeal_def by auto
    then have ¬(I·_I J⊆K) using assms
      using ideal_dest_subset[of I] ideal_dest_subset[of J] by auto
    moreover note K
    ultimately have K∈D(I·_I J) using openBasic_def[of I]
      ideal_dest_subset[OF assms(1)]
      unfolding Spec_def openBasic_def[OF S] by auto
  }
  then show D(I)∩D(J) ⊆ D(I·_I J) by auto
  {
    fix K assume Kass:K∈D(I·_I J)
    then have K:K◁_pR ¬(I·_I J⊆K) using openBasic_def[OF S] unfolding Spec_def
by auto
    {
      assume I ⊆ K ∨ J ⊆K
      then have I∩J ⊆ K by auto
      then have I·_I J⊆K using product_in_intersection assms by auto
      with K(2) have False by auto
    }
```

```
    then have ¬(I⊆K) ¬(J⊆K) by auto
    with Kass have K∈D(I)∩D(J) using assms ideal_dest_subset
      openBasic_def[of I] openBasic_def[of J]
      unfolding openBasic_def[OF S] Spec_def by auto
  }
  then show D(I ·I J) ⊆ D(I) ∩ D(J) by auto
qed
```

The union of D-sets is the D-set of the sum of the ideals

```
lemma (in ring0) union_open_basic:
  assumes 𝒥 ⊆ ℐ
  shows ⋃{D(I). I∈𝒥} = D(⊕I𝒥)
proof
  have S:(⊕I𝒥) ⊆ R using generated_ideal_is_ideal[of ⋃𝒥] assms
      unfolding sumArbitraryIdeals_def[OF assms]
      using ideal_dest_subset by auto
  {
    fix t assume t∈⋃{D(I). I∈𝒥}
    then obtain K where K:K∈𝒥 t∈D(K) by auto
    then have t:t◁pR ¬(K⊆t) using assms openBasic_def unfolding Spec_def
by auto
    {
      assume (⊕I𝒥) ⊆ t
      then have ⋃𝒥 ⊆ t using generated_ideal_contains_set[of ⋃𝒥]
        assms unfolding sumArbitraryIdeals_def[OF assms] by auto
      with K(1) have K ⊆ t by auto
      with t(2) have False by auto
    }
    then have ¬((⊕I𝒥) ⊆ t) by auto moreover
    note K S ultimately have t∈D(⊕I𝒥) using openBasic_def[of K] openBasic_def[of
⊕I𝒥]
      assms by auto
  }
  then show ⋃{D(I). I∈𝒥} ⊆ D(⊕I𝒥) by auto
  {
    fix t assume as:t∈D(⊕I𝒥)
    then have t:t∈Spec ¬((⊕I𝒥)⊆t) unfolding openBasic_def[OF S]
      by auto
    {
      assume ⋃𝒥 ⊆ t
      with t(1) have ⟨⋃𝒥⟩I ⊆ t using generated_ideal_small
        unfolding Spec_def primeIdeal_def
        by auto
      with t(2) have False unfolding sumArbitraryIdeals_def[OF assms]
        by auto
    }
    then obtain J where J:¬(J ⊆ t) J∈𝒥 by auto
    note J(1) moreover have J⊆R using `J∈𝒥` assms by auto
    moreover note t(1) ultimately have t∈D(J) using openBasic_def[of
```

```
J]
      by auto
    then have t:⋃{D(I). I∈𝒥} using J(2) by auto
  }
  then show D(⊕_I 𝒥) ⊆ ⋃{D(I). I∈𝒥} by auto
qed
```

From the previous results on intesertion and union, we conclude that the D-sets we computed form a topology

```
corollary (in ring0) zariski_top:
  shows {D(J). J∈ℐ}{is a topology} unfolding IsATopology_def
proof(safe)
  fix M assume M ⊆ {D(J). J∈ℐ}
  then have M = {D(J). J∈{K∈ℐ. D(K)∈M}} by auto
  then have ⋃M = ⋃{D(J). J∈{K∈ℐ. D(K)∈M}} by auto
  then have ⋃M = D(⊕_I{K∈ℐ. D(K)∈M}) using union_open_basic[of {K ∈
ℐ . D(K) ∈ M}] by auto
  moreover have (⊕_I{K∈ℐ. D(K)∈M})◁R using
    generated_ideal_is_ideal[of ⋃{K∈ℐ. D(K)∈M}]
    sumArbitraryIdeals_def [of {K∈ℐ. D(K)∈M}]
    by force
  then have (⊕_I{K∈ℐ. D(K)∈M})∈ℐ using ideal_dest_subset
    by auto
  ultimately show ⋃M:{D(J). J∈ℐ} by auto
next
  fix x xa assume as:x◁R xa◁R
  then have D(x) ∩ D(xa) = D(x·_I xa) using intersection_open_basic
    by auto
  moreover have (x·_I xa) ◁R using product_in_intersection(2)
    as by auto
  then have (x·_I xa):ℐ using ideal_dest_subset by auto
  ultimately show D(x) ∩ D(xa)∈{D(J). J∈ℐ} by auto
qed
```

We include all the results of topology0 into ring0 under the namespace "zariski"

```
definition(in ring0) ZarInt (int) where
  int(U) ≡ Interior(U,{D(J). J∈ℐ})

definition (in ring0) ZarCl (cl) where
  cl(U) ≡ Closure(U,{D(J). J∈ℐ})

definition (in ring0) ZarBound (∂_) where
  ∂U ≡ Boundary(U,{D(J). J∈ℐ})

sublocale ring0 < zariski:topology0 {D(J). J∈ℐ}
  ZarInt ZarCl ZarBound unfolding topology0_def
  ZarBound_def ZarInt_def ZarCl_def
  using zariski_top by auto
```

The interior of a proper subset is given by the D-set of the intersection of all the prime ideals not in that subset

```
lemma (in ring0) interior_zariski:
  assumes U ⊆ Spec U≠Spec
  shows int(U) = D(⋂(Spec-U))
proof
  {
    fix t assume t:t∈⋂(Spec-U)
    then have ∀V∈Spec-U. t:V by auto
    moreover from t have (Spec-U) ≠0 by auto
    ultimately obtain V where V∈Spec-U t∈V by auto
    then have t∈⋃Spec by auto
    then have t∈R unfolding Spec_def by auto
  }
  then have S:⋂(Spec-U) ⊆ R by auto
  {
    fix t assume t∈D(⋂(Spec-U))
    then have t:t:Spec ¬(⋂(Spec-U) ⊆ t) using openBasic_def[OF S]
      by auto
    {
      assume t∉U
      with t(1) have t∈Spec-U by auto
      then have ⋂(Spec-U) ⊆ t by auto
      then have False using t(2) by auto
    }
    then have t∈U by auto
  }
  then have D(⋂(Spec-U)) ⊆ U by auto moreover
  {
    assume Spec-U = 0
    with assms(1) have U=Spec by auto
    with assms(2) have False by auto
  }
  then have P:Spec-U ⊆ 𝓘 Spec-U ≠0 unfolding Spec_def by auto
  then have (⋂(Spec-U))◁R using intersection_ideals by auto
  then have (⋂(Spec-U))∈𝓘 using ideal_dest_subset by auto
  then have D(⋂(Spec-U)) ∈{D(J). J∈𝓘} by auto
  ultimately
  show D(⋂(Spec-U)) ⊆int(U) using zariski.Top_2_L5 by auto
  {
    fix V assume V:V∈{D(J). J∈𝓘} V ⊆ U
    from V(1) obtain J where J:J:𝓘 V=D(J) by auto
    from V(2) have SS:Spec-U ⊆ Spec-V by auto
    {
      fix K assume K:K∈Spec-U
      {
        assume ¬(J⊆K)
        with K have K∈D(J) using J(1) using openBasic_def
          by auto
```

```
      with SS K J(2) have False by auto
    }
    then have J⊆K by auto
  }
  then have J⊆ ⋂(Spec-U) using P(2) by auto
  then have D(J) ⊆ D(⋂(Spec-U)) using D_operator_preserve_subset
    S by auto
  with J(2) have V ⊆D(⋂(Spec-U)) by auto
}
then show int(U) ⊆ D(⋂(Spec-U)) using
  zariski.Top_2_L1 zariski.Top_2_L2 by auto
qed
```

The whole space is the D-set of the ring as an ideal of itself

```
lemma (in ring0) openBasic_total:
  shows D(R) = Spec
proof
  show D(R) ⊆ Spec using openBasic_def by auto
  {
    fix t assume t:t∈Spec
    {
      assume R ⊆ t
      then have False using t unfolding Spec_def primeIdeal_def
        using ideal_dest_subset[of t] by auto
    }
    with t have t∈D(R) using openBasic_def by auto
  }
  then show Spec ⊆ D(R) by auto
qed

corollary (in ring0) total_spec:
  shows ⋃{D(J). J∈ℐ} = Spec
proof
  show ⋃{D(J). J∈ℐ} ⊆ Spec using openBasic_def by auto
  have D(R)∈{D(J). J∈ℐ} using ring_self_ideal by auto
  then have D(R) ⊆ ⋃{D(J). J∈ℐ} by auto
  then show Spec ⊆ ⋃{D(J). J∈ℐ} using openBasic_total by auto
qed
```

The empty set is the D-set of the zero ideal

```
lemma (in ring0) openBasic_empty:
  shows D({0}) = 0
proof-
  {
    fix t assume t:t∈D({0})
    then have t◁pR ¬({0} ⊆ t) using openBasic_def
      Ring_ZF_1_L2(1) unfolding Spec_def by auto
    then have False using ideal_dest_zero unfolding primeIdeal_def by
auto
```

```
  }
  then show D({0}) =0 by auto
qed
```

A closed set is a set of primes containing a given ideal

```
lemma (in ring0) closeBasic:
  assumes U{is closed in}{D(J). J∈ℐ}
  obtains J where J∈ℐ and U={K∈Spec. J⊆K}
proof-
  assume rule:⋀J. J ∈ ℐ ⟹ U = {K ∈ Spec . J ⊆ K} ⟹ thesis
  from assms have U:U⊆Spec Spec-U ∈ {D(J). J∈ℐ} unfolding IsClosed_def
    using total_spec by auto
  then obtain J where J:J∈ℐ Spec-U = D(J) by auto
  moreover from U(1) have Spec-(Spec-U) = U by auto
  ultimately have U = Spec-{K∈Spec. ¬(J⊆K)} using openBasic_def
    by auto
  then have U = {K∈Spec. J⊆K} by auto
  with J show thesis using rule by auto
qed
```

We define the closed sets as V-sets

```
definition (in ring0) closeBasic (V) where
S ⊆ R ⟹ V(S) = {K∈Spec. S⊆K}
```

V-sets and D-sets are complementary

```
lemma (in ring0) V_is_closed:
  assumes J∈ℐ
  shows Spec-V(J) = D(J) and V(J){is closed in}{D(J). J∈ℐ}
  unfolding IsClosed_def
proof(safe)
  from assms have V(J) ⊆ Spec using closeBasic_def by auto
  then show ⋀x. x ∈ V(J) ⟹ x ∈ ⋃RepFun(ℐ, D) using total_spec by
auto
  show Spec-V(J) = D(J) using assms
    closeBasic_def openBasic_def by auto
  with assms show ⋃RepFun(ℐ, D) - V(J) ∈ RepFun(ℐ, D)
    using total_spec by auto
qed
```

As with D-sets, by De Morgan's Laws we get the same result for unions and intersections on V-sets

```
lemma (in ring0) V_union:
  assumes J∈ℐ K∈ℐ
  shows V(J)∪V(K) = V(J·ᵢK)
proof-
  {
    fix t assume t∈V(J)
    then have t∈Spec J⊆t using closeBasic_def
```

```
      assms(1) by auto
    moreover have J·_I K ⊆ J using product_in_intersection(1)[of J K]
      assms by auto
    ultimately have t∈Spec J·_I K ⊆t by auto
    then have t: V(J·_I K) using closeBasic_def
      product_in_intersection(2)[of J K] assms ideal_dest_subset
      by auto
  }
  moreover
  {
    fix t assume t∈V(K)
    then have t∈Spec K⊆t using closeBasic_def
      assms(2) by auto
    moreover have J·_I K ⊆ K using product_in_intersection(1)[of J K]
      assms by auto
    ultimately have t∈Spec J·_I K ⊆t by auto
    then have t: V(J·_I K) using closeBasic_def
      product_in_intersection(2)[of J K] assms ideal_dest_subset
      by auto
  }
  moreover
  {
    fix t assume t∈ V(J·_I K)
    then have t:t∈Spec J·_I K⊆t using closeBasic_def
      assms product_in_intersection(2)[of J K]
      ideal_dest_subset by auto
    from this(1) have ∀Ia∈ℐ. ∀J∈ℐ. Ia ·_I J ⊆ t ⟶ Ia ⊆ t ∨ J ⊆ t
      unfolding Spec_def primeIdeal_def by blast
    with assms have J·_I K⊆t ⟶ J⊆t ∨ K⊆t by auto
    with t have t∈Spec  J ⊆ t ∨ K ⊆ t by auto
    then have t∈V(J)∪V(K) using closeBasic_def
      assms by auto
  }
  ultimately show thesis by auto
qed

lemma (in ring0) V_intersect:
  assumes 𝒥 ⊆ ℐ 𝒥 ≠0
  shows ⋂{V(I). I∈𝒥} = V(⊕_I 𝒥)
proof-
  have Spec - (⋂{V(I). I∈𝒥}) = ⋃{D(I). I∈𝒥}
  proof
    {
      fix t assume t:Spec - (⋂{V(I). I∈𝒥})
      then have t:t:Spec t∉⋂{V(I). I∈𝒥} by auto
      from t(2) obtain K where (t∉V(K) ∧ K∈𝒥) ∨ 𝒥=0 by auto
      with assms(2) have t∉V(K) K∈𝒥 by auto
      with t(1) have t∈Spec-V(K) K:𝒥 by auto moreover
      from `K:𝒥` have Spec-V(K) = D(K) using assms(1) V_is_closed(1)
```

```
by auto
      ultimately have t:D(K) K:𝒥 by auto
      then have t∈⋃{D(I). I∈𝒥} by auto
    }
    then show Spec - (⋂{V(I). I∈𝒥}) ⊆ ⋃{D(I). I∈𝒥} by auto
    {
      fix t assume t∈⋃{D(I). I∈𝒥}
      then obtain K where K:K:𝒥 t:D(K) using assms(2)
        by auto
      from `K:𝒥` have Spec-V(K) = D(K) using assms(1) V_is_closed(1)
by auto
      with K(2) have t:Spec-V(K) by auto
      with K(1) have t∈Spec-⋂{V(I). I:𝒥} by auto
    }
    then show ⋃{D(I). I∈𝒥} ⊆ Spec - (⋂{V(I). I∈𝒥}) by auto
  qed
  then have Spec - (⋂{V(I). I∈𝒥}) = D(⊕I𝒥) using union_open_basic
    assms by auto
  then have Spec-(Spec - (⋂{V(I). I∈𝒥})) = Spec- D(⊕I𝒥) by auto
  moreover
  have JI:(⊕I𝒥) ∈ ℐ using generated_ideal_is_ideal[of ⋃𝒥] assms
    unfolding sumArbitraryIdeals_def[OF assms(1)]
    using ideal_dest_subset by auto
  then have Spec- V(⊕I𝒥) = D(⊕I𝒥) using V_is_closed(1)[of ⊕I𝒥]
    by auto
  then have Spec-(Spec-V(⊕I𝒥) ) = Spec-D(⊕I𝒥) by auto
  ultimately have R:Spec-(Spec - (⋂{V(I). I∈𝒥})) = Spec-(Spec-V(⊕I𝒥)
)
    by auto
  {
    fix t assume t:t∈⋂{V(I). I∈𝒥}
    with assms(2) obtain I where I:𝒥 t:V(I) by auto
    then have t∈Spec using closeBasic_def assms(1)
      by auto
    with t have t∈Spec-(Spec - (⋂{V(I). I∈𝒥})) by auto
    with R have t∈Spec-(Spec-V(⊕I𝒥) ) by auto
    then have t∈V(⊕I𝒥) by auto
  } moreover
  {
    fix t assume t:t∈V(⊕I𝒥)
    with JI have t:Spec using closeBasic_def by auto
    with t have t∈Spec-(Spec-V(⊕I𝒥) ) by auto
    then have t∈Spec-(Spec - (⋂{V(I). I∈𝒥})) using R by auto
    then have t∈⋂{V(I). I∈𝒥} by auto
  }
  ultimately show thesis by force
qed
```

The closure of a set is the V-set of the intersection of all its points.

```
lemma (in ring0) closure_zariski:
  assumes U ⊆ Spec U≠0
  shows cl(U) = V(⋂U)
proof
  have U ⊆ ℐ using assms(1) unfolding Spec_def by auto
  then have ideal:(⋂U)◁R using intersection_ideals[of U] assms(2) by
auto
  {
    fix t assume t:t∈V(⋂U)
    {
      fix q assume q:q∈⋂U
      with q obtain M where M:U q:M using assms(2) by blast
      with assms have q∈⋃Spec by auto
      then have q:R unfolding Spec_def by auto
    }
    then have sub:⋂U ⊆ R by auto
    with t have tt:t∈Spec ⋂U ⊆ t using closeBasic_def by auto
    {
      fix VV assume VV:VV∈{D(J). J∈ℐ} t∈VV
      then obtain J where J:VV= D(J) J∈ℐ by auto
      from VV(2) J have jt:¬(J ⊆ t) using openBasic_def by auto
      {
        assume U∩D(J) = 0
        then have ∀x∈U. x∉D(J) by auto
        with J(2) have ∀x∈U. J⊆x using openBasic_def[of J]
          assms(1) by auto
        then have J⊆ ⋂U ∨ U=0 by auto
        with tt(2) jt have False using assms(2) by auto
      }
      then have U∩VV ≠ 0 using J(1) by auto
    }
    then have R:∀VV∈{D(J). J∈ℐ}. t∈VV ⟶ VV∩U ≠ 0 by auto
    from tt(1) assms(1) have RR:t∈⋃{D(J). J∈ℐ} U ⊆ ⋃{D(J). J∈ℐ}
      using total_spec by auto
    have t∈cl(U) using zariski.inter_neigh_cl[OF RR(2,1) R].
  }
  then show V(⋂U) ⊆ cl(U)
    apply (rule subsetI) by auto
  {
    fix p assume p:p∈U
    then have ⋂U ⊆p by auto
    moreover
    from p assms(1) have p∈Spec ⋂U⊆⋃Spec by auto
    then have p∈Spec ⋂U⊆R unfolding Spec_def by auto
    ultimately have p∈V(⋂U) using closeBasic_def[of ⋂U]
      by auto
  }
  then have A:U⊆V(⋂U) by auto
  have B:V(⋂U){is closed in}{D(J). J∈ℐ}
```

```
    using V_is_closed(2) ideal ideal_dest_subset by auto
  show cl(U) ⊆ V(⋂U)
    apply (rule zariski.Top_3_L13[of V(⋂U) U])
    using A B by auto
qed

end
```

74 Rings - Zariski Topology - Properties

```
theory Ring_Zariski_ZF_2 imports Ring_Zariski_ZF Topology_ZF_1

begin

theorem (in ring0) zariski_t0:
  shows {D(I). I∈ℐ}{is T₀} unfolding isT0_def
proof-
  {
    fix x y assume ass:x:Spec y:Spec x≠y
    from ass(3) have ¬(x⊆y) ∨ ¬(y⊆x) by auto
    then have y∈D(x) ∨ x∈D(y) using ass(1,2)
      unfolding Spec_def using ass(1,2) openBasic_def
      by auto
    then have (x ∈ D(y) ∧ y ∉ D(y)) ∨ (y ∈ D(x) ∧ x ∉ D(x))
      using ass(1,2) unfolding Spec_def
      using openBasic_def by auto
    then have ∃U∈{D(I). I∈ℐ}. (x ∈ U ∧ y ∉ U) ∨ (y ∈ U ∧ x ∉ U)
      using ass(1,2) unfolding Spec_def by auto
  }
  then show ∀x y. x ∈ ⋃RepFun(ℐ, D) ∧ y ∈ ⋃RepFun(ℐ, D) ∧ x ≠ y ⟶
          (∃U∈RepFun(ℐ, D). x ∈ U ∧ y ∉ U ∨ y ∈ U ∧ x ∉ U)
    using total_spec by auto
qed
```

Noetherian rings have compact Zariski topology

```
theorem (in ring0) zariski_compact:
  assumes ∀I∈ℐ. (I{is finitely generated})
  shows Spec {is compact in} {D(I). I∈ℐ}
  unfolding IsCompact_def
proof(safe)
  show ⋀x. x ∈ Spec ⟹ x ∈ ⋃RepFun(ℐ, D) using total_spec by auto
  fix M assume M:M ⊆ RepFun(ℐ, D) Spec ⊆ ⋃M
  let J ={J∈ℐ. D(J)∈M}
  have m:M = RepFun(J,D) using M(1) by auto
  then have mm:⋃M = D(⊕ᵢJ) using union_open_basic[of J] by auto
  obtain T where T:T∈FinPow(J) (⊕ᵢJ) = ⊕ᵢT using
    sum_ideals_noetherian[OF assms(1), of J] by blast
  from T(2) have D(⊕ᵢJ) = D(⊕ᵢT) by auto
  moreover have T⊆ℐ using T(1) unfolding FinPow_def by auto
```

```
  ultimately have D(⊕_I J) = ⋃RepFun(T,D) using union_open_basic[of T]
    by auto
  with mm have ⋃M = ⋃RepFun(T,D) by auto
  then have Spec ⊆ ⋃RepFun(T,D) using M(2) by auto moreover
  from T(1) have RepFun(T,D) ⊆ RepFun(J,D) unfolding FinPow_def by auto
  with m have RepFun(T,D) ⊆ M by auto moreover
  from T(1) have Finite(RepFun(T,D)) unfolding FinPow_def
    using Finite_RepFun by auto
  ultimately show ∃N∈FinPow(M). Spec ⊆ ⋃N unfolding FinPow_def
    by auto
qed

end
```

75 Topology 1b

```
theory Topology_ZF_1b imports Topology_ZF_1

begin
```

One of the facts demonstrated in every class on General Topology is that in a T_2 (Hausdorff) topological space compact sets are closed. Formalizing the proof of this fact gave me an interesting insight into the role of the Axiom of Choice (AC) in many informal proofs.

A typical informal proof of this fact goes like this: we want to show that the complement of K is open. To do this, choose an arbitrary point $y \in K^c$. Since X is T_2, for every point $x \in K$ we can find an open set U_x such that $y \notin \overline{U_x}$. Obviously $\{U_x\}_{x\in K}$ covers K, so select a finite subcollection that covers K, and so on. I had never realized that such reasoning requires the Axiom of Choice. Namely, suppose we have a lemma that states "In T_2 spaces, if $x \neq y$, then there is an open set U such that $x \in U$ and $y \notin \overline{U}$" (like our lemma `T2_cl_open_sep` below). This only states that the set of such open sets U is not empty. To get the collection $\{U_x\}_{x\in K}$ in this proof we have to select one such set among many for every $x \in K$ and this is where we use the Axiom of Choice. Probably in 99/100 cases when an informal calculus proof states something like $\forall \varepsilon \exists \delta_\varepsilon \cdots$ the proof uses AC. Most of the time the use of AC in such proofs can be avoided. This is also the case for the fact that in a T_2 space compact sets are closed.

75.1 Compact sets are closed - no need for AC

In this section we show that in a T_2 topological space compact sets are closed.

First we prove a lemma that in a T_2 space two points can be separated by the closure of an open set.

```
lemma (in topology0) T2_cl_open_sep:
  assumes T {is T₂}  and x ∈ ⋃T  y ∈ ⋃T  x≠y
  shows ∃U∈T. (x∈U ∧ y ∉ cl(U))
proof -
  from assms have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0
    using isT2_def by simp
  then obtain U V where U∈T  V∈T  x∈U  y∈V  U∩V=0
    by auto
  then have U∈T ∧ x∈U ∧ y∈ V ∧ cl(U) ∩ V = 0
    using  disj_open_cl_disj by auto
  thus ∃U∈T. (x∈U ∧ y ∉ cl(U)) by auto
qed
```

AC-free proof that in a Hausdorff space compact sets are closed. To understand the notation recall that in Isabelle/ZF Pow(A) is the powerset (the set of subsets) of A and FinPow(A) denotes the set of finite subsets of A in IsarMathLib.

```
theorem (in topology0) in_t2_compact_is_cl:
  assumes A1: T {is T₂} and A2: K {is compact in} T
  shows K {is closed in} T
proof -
  let X = ⋃T
  have ∀y ∈ X - K. ∃U∈T. y∈U ∧ U ⊆ X - K
  proof -
    { fix y assume y ∈ X  y∉K
      have ∃U∈T. y∈U ∧ U ⊆ X - K
      proof -
 let B = ⋃x∈K. {V∈T. x∈V ∧ y ∉ cl(V)}
 have I: B ∈ Pow(T)  FinPow(B) ⊆ Pow(B)
   using FinPow_def by auto
 from ‹K {is compact in} T› ‹y ∈ X›  ‹y∉K› have
   ∀x∈K. x ∈ X ∧ y ∈ X ∧ x≠y
   using IsCompact_def by auto
 with ‹T {is T₂}› have ∀x∈K. {V∈T. x∈V ∧ y ∉ cl(V)} ≠ 0
   using T2_cl_open_sep by auto
 hence K ⊆ ⋃B by blast
 with ‹K {is compact in} T› I have
   ∃N ∈ FinPow(B). K ⊆ ⋃N
   using IsCompact_def by auto
 then obtain N where N ∈ FinPow(B)  K ⊆ ⋃N
   by auto
 with I have N ⊆ B by auto
 hence ∀V∈N. V∈B by auto
 let M = {cl(V). V∈N}
 let C = {D ∈ Pow(X). D {is closed in} T}
 from ‹N ∈ FinPow(B)› have ∀V∈B. cl(V) ∈ C  N ∈ FinPow(B)
   using cl_is_closed IsClosed_def by auto
 then have M ∈ FinPow(C) by (rule fin_image_fin)
 then have X - ⋃M ∈ T using fin_union_cl_is_cl IsClosed_def
```

```
    by simp
  moreover from ‹y ∈ X›  ‹y∉K›  ‹∀V∈N. V∈B› have
    y ∈ X - ⋃M by simp
  moreover have X - ⋃M ⊆ X - K
  proof -
    from ‹∀V∈N. V∈B› have ⋃N ⊆ ⋃M using cl_contains_set by auto
    with ‹K ⊆ ⋃N› show X - ⋃M ⊆ X - K by auto
  qed
  ultimately have ∃U. U∈T ∧ y ∈ U ∧ U ⊆ X - K
    by auto
  thus ∃U∈T. y∈U ∧ U ⊆ X - K by auto
      qed
    } thus ∀y ∈ X - K. ∃U∈T. y∈U ∧ U ⊆ X - K
      by auto
  qed
  with A2 show K {is closed in} T
    using open_neigh_open IsCompact_def IsClosed_def by auto
qed


end
```

76 Rings - Zariski Topology - maps

```
theory Ring_Zariski_ZF_3 imports Ring_Zariski_ZF Ring_ZF_3 Topology_ZF_2

begin

lemma (in ring_homo) spectrum_surj:
  defines g ≡ λu∈target_ring.Spec. f-u
  assumes f∈surj(R,S)
  shows g: target_ring.Spec → V(ker)
proof-
  have g: target_ring.Spec → {f-u. u∈target_ring.Spec} using lam_funtype
    unfolding g_def by auto moreover
  {
    fix t assume t∈{f-u. u∈target_ring.Spec}
    then obtain u where u:u:target_ring.Spec t=f-u by auto
    from u(1) have u2:u◁_pR_t unfolding target_ring.Spec_def by auto
    then have (f-u)◁_pR_o using preimage_prime_ideal_surj
      assms(2) by auto moreover
    then have f-u ∈ origin_ring.ideals
      unfolding origin_ring.primeIdeal_def
      using origin_ring.ideal_dest_subset by auto
    ultimately have f-u ∈ origin_ring.Spec unfolding origin_ring.Spec_def
      by auto
    moreover from u2 have 0_S ∈ u unfolding
      target_ring.primeIdeal_def using target_ring.ideal_dest_zero
      by auto
```

```
    then have {0_S} ⊆ u by auto
    then have f-{0_S} ⊆ f-u by auto
    moreover have f-u ⊆ R using func1_1_L15[OF
      surj_is_fun[OF assms(2)]] by auto
    ultimately have f-u ∈ origin_ring.closeBasic(f-{0_S})
      using origin_ring.closeBasic_def[of f-{0_S}]
      by force
    with u(2) have t∈origin_ring.closeBasic(f-{0_S}) by auto
  }
  then have {f-u. u∈target_ring.Spec} ⊆ origin_ring.closeBasic(f-{0_S})
by auto
  ultimately show thesis using func1_1_L1B by auto
qed

lemma (in ring_homo) spectrum_surj_bij:
  defines g ≡ λu∈target_ring.Spec. f-u
  assumes f∈surj(R,S)
  shows g∈bij(target_ring.Spec, V(ker))
proof-
  {
    fix s t assume st:s∈target_ring.Spec t∈target_ring.Spec
      gs = gt
    then have f-s = f-t using beta unfolding g_def by auto
    then have f(f-s) = f(f-t) by auto
    moreover from st(1,2) have s ⊆ S t ⊆S
      unfolding target_ring.Spec_def origin_ring.Spec_def
      by auto
    moreover note assms(2) st(1,2)
    ultimately have s=t using surj_image_vimage
      unfolding target_ring.Spec_def origin_ring.Spec_def
      by auto
  }
  then have g∈inj(target_ring.Spec, origin_ring.closeBasic(f-{0_S}))
    unfolding inj_def using spectrum_surj assms(2) unfolding g_def by
auto
  moreover
  {
    fix t assume t:t∈origin_ring.closeBasic(f-{0_S})
    then have tt:t∈origin_ring.Spec f-{0_S} ⊆ t
      using origin_ring.closeBasic_def func1_1_L6A[OF surj_is_fun]
      assms(2) by auto
    {
      fix y assume y:y∈f-(ft)
      then have y:y∈R fy∈ft using func1_1_L15
        surj_is_fun[OF assms(2)] by auto
      from y(2) obtain x where x:x∈t fy = fx
        using func_imagedef[OF surj_is_fun]
        assms(2) tt(1) unfolding origin_ring.Spec_def by auto
      from x(2) have (fy)-_S(fx) = (fx)-_S(fx) by auto
```

```
      then have (fy)-S(fx) = 0S using target_ring.Ring_ZF_1_L3(7)
        apply_type[OF surj_is_fun] assms(2) x(1) tt(1)
        unfolding origin_ring.Spec_def by auto
      then have f(y-Rx) = 0S using homomor_dest_subs
        x(1) tt(1) y(1) unfolding origin_ring.Spec_def by auto more-
over
      from x(1) tt(1) have x∈R unfolding origin_ring.Spec_def by auto
      with y(1) have y-Rx ∈R using origin_ring.Ring_ZF_1_L4(2) by auto
      ultimately have y-Rx ∈ f-{0S} using func1_1_L15
        surj_is_fun[OF assms(2)] by auto
      then have y-Rx ∈t using tt(2) by auto moreover
      have t◁Ro using tt(1) unfolding origin_ring.Spec_def by auto
      ultimately have x+R(y-Rx) ∈t using x(1)
        origin_ring.ideal_dest_sum by auto
      then have y∈t using origin_ring.Ring_ZF_2_L1A(5)
        y(1) ‹x∈R› by auto
    }
    then have f-(ft) = t using func1_1_L9[of f R S t]
      surj_is_fun[OF assms(2)] tt(1) unfolding origin_ring.Spec_def
      by auto moreover
    then have (ft)◁pRt using prime_ideal_quot_3[of ft]
      assms(2) tt(1) unfolding origin_ring.Spec_def
      using image_ideal_surj[of t] origin_ring.primeIdeal_def[of t]
      by auto
    then have (ft)◁pRt (ft)◁Rt
      using target_ring.primeIdeal_def[of ft]
      by auto
    then have (ft):target_ring.Spec
      unfolding target_ring.Spec_def using target_ring.ideal_dest_subset
      by auto moreover
    then have f-(ft) = g(ft) unfolding g_def
      using beta by auto
    ultimately have g(ft) = t (ft):target_ring.Spec by auto
    then have ∃p∈target_ring.Spec. gp = t by auto
  }
  ultimately show g:bij(target_ring.Spec, V(ker))
    unfolding bij_def surj_def inj_def by auto
qed

definition (in ring_homo) top_origin (τo) where
  top_origin ≡ {origin_ring.openBasic(J) . J ∈ origin_ring.ideals}

definition (in ring_homo) top_target (τt) where
  top_target ≡ {target_ring.openBasic(J) . J ∈ target_ring.ideals}

definition (in ring_homo) spec_cont where
 spec_cont(h) ≡ IsContinuous(τt, τo, h)

lemma (in ring_homo) spectrum_surj_cont:
```

```
  defines g ≡ λu∈target_ring.Spec. f-u
  assumes f∈surj(R,S)
  shows IsContinuous(τt, τo {restricted to}(V(ker)), g)
  unfolding IsContinuous_def top_target_def RestrictedTo_def top_origin_def
proof(safe)
  fix x assume ass:x◁Ro x ⊆ R
  have origin_ring.openBasic(x) = {u∈origin_ring.Spec. ¬(x ⊆ u)}
    unfolding origin_ring.openBasic_def[OF ass(2)] by auto
  have g-(origin_ring.openBasic(x)) = {t∈target_ring.Spec. gt ∈ origin_ring.openBasic(x)}

    using spectrum_surj assms(2) unfolding g_def
    using func1_1_L15 by auto
  then have G:g-(origin_ring.openBasic(x)) = {t∈target_ring.Spec. f-t
∈ origin_ring.openBasic(x)}
    using beta unfolding g_def by auto
  have (fx)◁Rt using image_ideal_surj assms(2) ass(1) by auto
  then have H:fx∈target_ring.ideals using
    target_ring.ideal_dest_subset by auto
  then have F:target_ring.openBasic(fx) = {t∈target_ring.Spec. ¬(fx ⊆
t)}
    using target_ring.openBasic_def by auto
  {
    fix s assume s∈{t∈target_ring.Spec. f-t ∈ origin_ring.openBasic(x)}
    then have s:s∈target_ring.Spec f-s ∈ origin_ring.openBasic(x)
      by auto
    from this(2) have E:f-s ∈origin_ring.Spec ¬(x ⊆ f-s)
      using origin_ring.openBasic_def ass(1) origin_ring.ideal_dest_subset
      by auto
    {
      assume fx ⊆ s
      then have f-(fx) ⊆ f-s by auto
      then have x ⊆ f-s using func1_1_L9[OF surj_is_fun]
        assms(2) ass(1) origin_ring.ideal_dest_subset by force
      with E(2) have False by auto
    }
    then have ¬(fx ⊆ s) by auto
    with s(1) have s∈{t∈target_ring.Spec. ¬(fx ⊆ t)}
      by auto
  }
  then have {t ∈ target_ring.Spec . f - t ∈ origin_ring.openBasic(x)}
⊆ {t∈target_ring.Spec. ¬(fx ⊆ t)}
    by auto moreover
  {
    fix s assume s∈{t∈target_ring.Spec. ¬(fx ⊆ t)}
    then have s:s∈target_ring.Spec ¬(fx ⊆ s) by auto
    have origin_ring.openBasic(x) = {t∈origin_ring.Spec. ¬(x ⊆ t)}
      using origin_ring.openBasic_def ass(1) origin_ring.ideal_dest_subset
      by auto
    {
```

```
      assume x ⊆ f-s
      then have fx ⊆ f(f-s) by auto
      then have fx ⊆ s using surj_image_vimage
        assms(2) s(1) unfolding target_ring.Spec_def
          target_ring.primeIdeal_def target_ring.ideal_dest_subset
        by auto
      with s(2) have False by auto
    }
    then have ¬(x ⊆ f-s) by auto
    moreover
    from s(1) have (f-s) ◁pRo unfolding target_ring.Spec_def
      using preimage_prime_ideal_surj assms(2) by auto
    then have (f-s)∈origin_ring.Spec unfolding origin_ring.Spec_def
      origin_ring.primeIdeal_def using origin_ring.ideal_dest_subset
      by auto
    ultimately have f-s∈origin_ring.openBasic(x)
      using origin_ring.openBasic_def ass(1)
      origin_ring.ideal_dest_subset by auto
    then have s∈{t ∈ target_ring.Spec . f - t ∈ origin_ring.openBasic(x)}
      using s(1) by auto
  }
  then have {t ∈ target_ring.Spec . ¬ f  x ⊆ t} ⊆ {t ∈ target_ring.Spec
. f - t ∈ origin_ring.openBasic(x)}
    by auto
  ultimately have {t ∈ target_ring.Spec . ¬ f  x ⊆ t} = {t ∈ target_ring.Spec
. f - t ∈ origin_ring.openBasic(x)}
    by auto
  with F have target_ring.openBasic(fx) = {t ∈ target_ring.Spec . f -
t ∈ origin_ring.openBasic(x)}
    by auto
  with G have T:target_ring.openBasic(fx) = g- origin_ring.openBasic(x)
by auto
  have g-(origin_ring.closeBasic(f - {0S})) = {t∈target_ring.Spec. gt
∈ origin_ring.closeBasic(f - {0S})}
    using spectrum_surj assms(2) unfolding g_def
    using func1_1_L15 by auto
  then have g-(origin_ring.closeBasic(f - {0S})) = {t∈target_ring.Spec.
f-t ∈ origin_ring.closeBasic(f - {0S})}
    using beta unfolding g_def by auto
  then have E:g-(origin_ring.closeBasic(f - {0S})) = {t∈target_ring.Spec.
f-t ∈ {q∈origin_ring.Spec. (f - {0S}) ⊆ q}}
    unfolding origin_ring.closeBasic_def[OF func1_1_L3[OF surj_is_fun[OF
assms(2)]]] by auto
  {
    fix s assume s:s∈target_ring.openBasic(fx)
    with E have ss:s∈target_ring.Spec ¬(fx ⊆ s)
      using target_ring.openBasic_def func1_1_L6(2) surj_is_fun[OF assms(2)]
by auto
    from this(1) have gs ∈ origin_ring.closeBasic(f - {0S}) using spectrum_surj[OF
```

```
assms(2)]
      apply_type[of g target_ring.Spec λu. origin_ring.closeBasic(f -
{0_S})] unfolding g_def
      by auto
    with ss(1) have f-s ∈ origin_ring.closeBasic(f - {0_S}) using beta
      unfolding g_def by auto
    moreover
    from ss(1) have s◁R_t unfolding target_ring.Spec_def
      target_ring.primeIdeal_def by auto
    then have 0_S ∈s using target_ring.ideal_dest_zero by auto
    then have {0_S} ⊆ s by auto
    then have f-{0_S} ⊆ f-s by auto
    moreover
    have f-{0_S} ⊆ R using func1_1_L15[OF
      surj_is_fun[OF assms(2)], of {0_S}] by auto
    ultimately have f-s ∈ {q∈origin_ring.Spec. (f - {0_S}) ⊆ q}
      using origin_ring.closeBasic_def by auto
    then have s∈{t∈target_ring.Spec. f-t ∈ {q∈origin_ring.Spec. (f -
{0_S}) ⊆ q}}
      using ss(1) by auto
    then have s∈g-(origin_ring.closeBasic(f - {0_S})) using E by auto
  }
  then have target_ring.openBasic(fx) ⊆ g-(origin_ring.closeBasic(f -
{0_S})) by auto
  then have g-(origin_ring.closeBasic(f - {0_S}))∩target_ring.openBasic(fx)
= target_ring.openBasic(fx)
    by auto
  with T have g-(origin_ring.closeBasic(f - {0_S})) ∩ g - origin_ring.openBasic(x)
= target_ring.openBasic(fx)
    by auto moreover
  have g-(origin_ring.closeBasic(f - {0_S})) ∩ g - origin_ring.openBasic(x)
=
    g-(origin_ring.closeBasic(f - {0_S}) ∩ origin_ring.openBasic(x))
    using invim_inter_inter_invim[OF spectrum_surj[OF assms(2)]]
    unfolding g_def by auto
  ultimately have g -
      (origin_ring.closeBasic(f - {0_S}) ∩
       origin_ring.openBasic(x)) = target_ring.openBasic(fx)
    by auto
  with H show g -
      (origin_ring.closeBasic(f - {0_S}) ∩
       origin_ring.openBasic(x)) ∈
          RepFun(target_ring.ideals, target_ring.openBasic)
    by auto
qed

lemma (in ring_homo) spectrum_surj_open:
  defines g ≡ λu∈target_ring.Spec. f-u
  assumes f∈surj(R,S)
```

```
  shows ∀U∈τt. gU ∈ τo {restricted to} V(ker)
proof
  fix U assume U:U∈τt
  then obtain I where I:I◁Rt I⊆S
    U=target_ring.openBasic(I) unfolding top_target_def
    by auto
  from I(3) have sub:U ⊆ target_ring.Spec
    using target_ring.openBasic_def[OF I(2)] by auto
  {
    fix t assume t:t∈gU
    then obtain u where u:u∈U t=gu
      using func_imagedef spectrum_surj[OF assms(2)] sub
      unfolding g_def by auto
    then have t:t=f-u using beta sub unfolding g_def by auto
    with sub u(1) have t◁pRo unfolding target_ring.Spec_def
      using preimage_prime_ideal_surj[OF _ assms(2), of u]
      by auto
    then have p:t∈origin_ring.Spec unfolding origin_ring.Spec_def
      unfolding origin_ring.primeIdeal_def
      using origin_ring.ideal_dest_subset by auto
    from u(1) I(2,3) have Iu:¬ (I ⊆ u) using target_ring.openBasic_def
      by auto
    {
      assume f-I ⊆ t
      then have f(f-I) ⊆ ft by auto
      then have I ⊆ ft using surj_image_vimage[OF assms(2)] I(2) by auto
      with t have I ⊆ f(f-u) by auto
      moreover from u(1) sub have u ⊆ S
        unfolding target_ring.Spec_def by auto
      ultimately have I ⊆ u using surj_image_vimage[OF assms(2)]
        by auto
      with Iu have False by auto
    }
    then have ¬(f-I ⊆ t) by auto
    with p have t∈origin_ring.openBasic(f-I)
      using origin_ring.openBasic_def func1_1_L6A[OF surj_is_fun]
      assms(2) by auto
  }
  then have gU ⊆ origin_ring.openBasic(f-I) by auto moreover
  {
    fix t assume t:t∈origin_ring.openBasic(f-I) t∈V(ker)
    have f-I ⊆ R using func1_1_L6A[OF surj_is_fun]
      assms(2) by auto
    with t have p:t∈origin_ring.Spec ¬(f-I ⊆ t)
      using origin_ring.openBasic_def by auto
    from t(2) have kt:ker ⊆ t using origin_ring.closeBasic_def
      func1_1_L3 f_is_fun by auto
    {
      fix x assume x∈f-(ft)
```

```
      then have t:fx∈ft x∈R using func1_1_L15
        surj_is_fun[OF assms(2)] by auto
      then obtain s where s:fx = fs s∈t using
        func_imagedef[OF surj_is_fun[OF assms(2)]]
        p(1) unfolding origin_ring.Spec_def by auto
      from s(2) have ss:s∈R using p(1)
        unfolding origin_ring.Spec_def by auto
      from s(1) have (fx) -S (fs) = 0S using
        target_ring.Ring_ZF_1_L3(7)[OF apply_type[OF
            surj_is_fun[OF assms(2)]
        t(2)]] by auto
      then have f(x-Rs) = 0S using homomor_dest_subs
        t(2) ss by auto moreover
      from t(2) ss have x-Rs ∈R using origin_ring.Ring_ZF_1_L4(2) by
auto
      ultimately have x-Rs ∈ f-{0S} using func1_1_L15
        surj_is_fun[OF assms(2)] by auto
      then have x-Rs ∈ t using kt by auto
      then have s+R(x-Rs) ∈ t
        using origin_ring.ideal_dest_sum
        s(2) p(1) unfolding origin_ring.Spec_def by auto
      then have x∈t using origin_ring.Ring_ZF_2_L1A(5)
        ss t(2) by auto
    }
    then have eq:f-(ft) = t
      using func1_1_L9[OF surj_is_fun[OF assms(2)]
      origin_ring.ideal_dest_subset[of t]] p(1) unfolding origin_ring.Spec_def
      origin_ring.primeIdeal_def by auto
    then have (f  t)◁Rt ⟹ (ft)◁pRt
      using prime_ideal_quot_3[of ft] assms(2)
      p(1) unfolding origin_ring.Spec_def by auto
    then have id:(ft)◁pRt (f  t)◁Rt using image_ideal_surj
      p(1) assms(2) unfolding origin_ring.Spec_def by auto
    {
      assume I ⊆ ft
      then have f-I ⊆ f-(ft) by auto
      with eq have f-I ⊆ t by auto
      with p(2) have False by auto
    }
    then have ¬(I ⊆ ft) by auto
    then have ft∈target_ring.openBasic(I)
      using id target_ring.ideal_dest_subset unfolding target_ring.openBasic_def[OF
I(2)]
      target_ring.Spec_def by auto
    then have q:ft ∈U using I(3) by auto
    then have q2:ft∈target_ring.Spec using sub by auto
    from q have g(ft) ∈gU using func1_1_L15D[OF bij_is_fun
      [OF spectrum_surj_bij[OF assms(2)]], of ft U]
      unfolding g_def using sub by auto
```

```
    then have f-(ft) ∈ gU using beta[of ft
      target_ring.Spec λo. f-o] q2
      unfolding g_def by auto
    with eq have t∈gU by auto
  }
  then have V(ker)∩D(f - I) ⊆ gU by auto
  ultimately
  have V(ker)∩D(f - I) = gU
    using func1_1_L6(2)[OF bij_is_fun[OF
    spectrum_surj_bij[OF assms(2)]],of U]
    unfolding g_def by blast
  moreover
  from I(1) have (f-(I)) ◁R and (f-(I)) ⊆ R
    using preimage_ideal(2)  origin_ring.ideal_dest_subset by simp_all
  then have V(ker)∩D(f-(I)) ∈ {V(ker) ∩ A . A ∈ τ_o}
    unfolding top_origin_def by auto
  ultimately show gU:  τ_o{restricted to}V(ker)
    unfolding RestrictedTo_def by auto
qed
```

A quotient ring has a spectrum homeomorphic to a closed subspace of the spectrum of the base ring. Specifically, the closed subspace associated to the ideal by which we quotient.

```
corollary (in ring_homo) surj_homeomorphism:
  assumes f∈surj(R,S)
  defines g ≡ λu∈target_ring.Spec. f - u
  shows IsAhomeomorphism(τ_t, τ_o{restricted to}V(ker), g)
proof-
  have ⋃(τ_o{restricted to}V(ker)) = origin_ring.Spec ∩ V(ker) unfold-
ing
    top_origin_def RestrictedTo_def using origin_ring.total_spec
    by auto
  then have ⋃(τ_o{restricted to}V(ker)) = V(ker)
    using origin_ring.closeBasic_def func1_1_L3 f_is_fun
     by auto moreover
  have ⋃τ_t = target_ring.Spec unfolding top_target_def
    using target_ring.total_spec by auto
  ultimately show thesis using bij_cont_open_homeo[of g τ_t τ_o{restricted
to}V(ker)]
    spectrum_surj_bij[OF assms(1)] spectrum_surj_open[OF assms(1)]
    spectrum_surj_cont[OF assms(1)]
    unfolding g_def by auto
qed

end
```

77 Topology 3

theory `Topology_ZF_3` **imports** `Topology_ZF_2 FiniteSeq_ZF`

begin

`Topology_ZF_1` theory describes how we can define a topology on a product of two topological spaces. One way to generalize that is to construct topology for a cartesian product of n topological spaces. The cartesian product approach is somewhat inconvenient though. Another way to approach product topology on X^n is to model cartesian product as sets of sequences (of length n) of elements of X. This means that having a topology on X we want to define a toplogy on the space $n \to X$, where n is a natural number (recall that $n = \{0, 1, ..., n-1\}$ in ZF). However, this in turn can be done more generally by defining a topology on any function space $I \to X$, where I is any set of indices. This is what we do in this theory.

77.1 The base of the product topology

In this section we define the base of the product topology.

Suppose $\mathcal{X} = I \to \bigcup T$ is a space of functions from some index set I to the carrier of a topology T. Then take a finite collection of open sets $W : N \to T$ indexed by $N \subseteq I$. We can define a subset of $\mathcal{X}$ that models the cartesian product of W.

```
definition
  FinProd(𝒳,W) ≡ {x∈𝒳. ∀ i∈domain(W). x(i) ∈ W(i)}
```

Now we define the base of the product topology as the collection of all finite products (in the sense defined above) of open sets.

```
definition
  ProductTopBase(I,T) ≡ ⋃N∈FinPow(I).{FinProd(I→⋃T,W). W∈N→T}
```

Finally, we define the product topology on sequences. We use the "Seq" prefix although the definition is good for any index sets, not only natural numbers.

```
definition
  SeqProductTopology(I,T) ≡ {⋃B. B ∈ Pow(ProductTopBase(I,T))}
```

Product topology base is closed with respect to intersections.

```
lemma prod_top_base_inter:
  assumes A1: T {is a topology} and
  A2: U ∈ ProductTopBase(I,T)  V ∈ ProductTopBase(I,T)
  shows U∩V ∈ ProductTopBase(I,T)
proof -
  let 𝒳 = I→⋃T
```

```
  from A2 obtain N1  W1 N2 W2 where
    I: N1 ∈ FinPow(I)  W1∈N1→T  U = FinProd(𝒳,W1) and
    II: N2 ∈ FinPow(I)  W2∈N2→T  V = FinProd(𝒳,W2)
    using ProductTopBase_def by auto
  let N3 = N1 ∪ N2
  let W3 = {⟨i,if i ∈ N1-N2 then W1(i)
       else if i ∈ N2-N1 then  W2(i)
       else (W1(i)) ∩ (W2(i))⟩. i ∈ N3}
  from A1 I II have ∀i ∈ N1 ∩ N2.  (W1(i) ∩ W2(i)) ∈ T
      using apply_funtype IsATopology_def by auto
  moreover from I II have ∀i∈N1-N2. W1(i) ∈ T and ∀i∈N2-N1. W2(i) ∈ T
      using apply_funtype by auto
  ultimately have  W3:N3→T by (rule fun_union_overlap)
  with I II have FinProd(𝒳,W3) ∈ ProductTopBase(I,T) using union_finpow ProductTopBase_def
    by auto
  moreover have U∩V = FinProd(𝒳,W3)
  proof
    { fix x assume x∈U and x∈V
      with ‹U = FinProd(𝒳,W1)›  ‹W1∈N1→T› and  ‹V = FinProd(𝒳,W2)› ‹W2∈N2→T›
      have x∈𝒳 and ∀i∈N1. x(i) ∈ W1(i) and ∀i∈N2. x(i) ∈ W2(i)
        using func1_1_L1 FinProd_def by auto
      with ‹W3:N3→T› ‹x∈𝒳› have x ∈ FinProd(𝒳,W3)
        using ZF_fun_from_tot_val func1_1_L1 FinProd_def by auto
    } thus U∩V ⊆ FinProd(𝒳,W3) by auto
    { fix x assume x ∈ FinProd(𝒳,W3)
      with ‹W3:N3→T› have x:I→⋃T and III: ∀i∈N3. x(i) ∈ W3(i)
        using FinProd_def func1_1_L1 by auto
     { fix i assume i∈N1
       with ‹W3:N3→T› have W3(i) ⊆ W1(i) using ZF_fun_from_tot_val by auto
       with III ‹i∈N1› have x(i) ∈ W1(i) by auto
     } with ‹W1∈N1→T› ‹x:I→⋃T› ‹U = FinProd(𝒳,W1)›
      have x ∈ U using func1_1_L1 FinProd_def by auto
      moreover
      { fix i assume i∈N2
        with ‹W3:N3→T› have W3(i) ⊆ W2(i) using ZF_fun_from_tot_val by auto
        with III ‹i∈N2› have x(i) ∈ W2(i) by auto
      } with ‹W2∈N2→T› ‹x:I→⋃T› ‹V = FinProd(𝒳,W2)› have x∈V
          using func1_1_L1 FinProd_def by auto
      ultimately have x ∈ U∩V by simp
    } thus FinProd(𝒳,W3) ⊆ U∩V by auto
  qed
    ultimately show thesis by simp
qed
```

In the next theorem we show the collection of sets defined above as `ProductTopBase(𝒳,T)` satisfies the base condition. This is a condition, defined in `Topology_ZF_1` that allows to claim that this collection is a base for some topology.

```
theorem prod_top_base_is_base: assumes T {is a topology}
  shows ProductTopBase(I,T) {satisfies the base condition}
  using assms prod_top_base_inter inter_closed_base by simp
```

The (sequence) product topology is indeed a topology on the space of sequences. In the proof we are using the fact that $(\emptyset \to X) = \{\emptyset\}$.

```
theorem seq_prod_top_is_top:  assumes T {is a topology}
  shows
  SeqProductTopology(I,T) {is a topology} and
  ProductTopBase(I,T) {is a base for} SeqProductTopology(I,T) and
  ⋃SeqProductTopology(I,T) = (I→⋃T)
proof -
  from assms show SeqProductTopology(I,T) {is a topology} and
    I: ProductTopBase(I,T) {is a base for} SeqProductTopology(I,T)
      using prod_top_base_is_base SeqProductTopology_def Top_1_2_T1
        by auto
  from I have ⋃SeqProductTopology(I,T) = ⋃ProductTopBase(I,T)
    using Top_1_2_L5 by simp
  also have ⋃ProductTopBase(I,T) = (I→⋃T)
  proof
    show ⋃ProductTopBase(I,T) ⊆ (I→⋃T) using ProductTopBase_def FinProd_def
      by auto
    have 0 ∈ FinPow(I) using empty_in_finpow by simp
    hence {FinProd(I→⋃T,W). W∈0→T} ⊆ (⋃N∈FinPow(I).{FinProd(I→⋃T,W).
W∈N→T})
      by blast
    then show (I→⋃T) ⊆ ⋃ProductTopBase(I,T) using ProductTopBase_def
FinProd_def
      by auto
  qed
  finally show ⋃SeqProductTopology(I,T) = (I→⋃T) by simp
qed
```

77.2 Finite product of topologies

As a special case of the space of functions $I \to X$ we can consider space of lists of elements of X, i.e. space $n \to X$, where n is a natural number (recall that in ZF set theory $n = \{0, 1, ..., n-1\}$). Such spaces model finite cartesian products X^n but are easier to deal with in formalized way (than the said products). This section discusses natural topology defined on $n \to X$ where X is a topological space.

When the index set is finite, the definition of `ProductTopBase(I,T)` can be simplifed.

```
lemma fin_prod_def_nat: assumes A1: n∈nat and A2: T {is a topology}
```

```
  shows ProductTopBase(n,T) = {FinProd(n→⋃T,W). W∈n→T}
proof
  from A1 have n ∈ FinPow(n) using nat_finpow_nat fin_finpow_self by
auto
  then show {FinProd(n→⋃T,W). W∈n→T} ⊆ ProductTopBase(n,T) using ProductTopBase_def
    by auto
  { fix B assume B ∈ ProductTopBase(n,T)
    then obtain N W where  N ∈ FinPow(n) and W ∈ N→T and B = FinProd(n→⋃T,W)
      using ProductTopBase_def by auto
    let Wn = {⟨i,if i∈N then W(i) else ⋃T⟩. i∈n}
    from A2  ‹N ∈ FinPow(n)›  ‹W∈N→T› have ∀i∈n. (if i∈N then W(i)
else ⋃T) ∈ T
      using apply_funtype FinPow_def IsATopology_def by auto
    then have Wn:n→T by (rule ZF_fun_from_total)
    moreover have B = FinProd(n→⋃T,Wn)
    proof
      { fix x assume x∈B
        with ‹B = FinProd(n→⋃T,W)› have x ∈ n→⋃T using FinProd_def
by simp
        moreover have ∀i∈domain(Wn). x(i) ∈ Wn(i)
        proof
          fix i assume i ∈ domain(Wn)
          with ‹Wn:n→T› have i∈n using func1_1_L1 by simp
          with ‹x:n→⋃T› have x(i) ∈ ⋃T using apply_funtype by blast
          with ‹x∈B› ‹B = FinProd(n→⋃T,W)› ‹W ∈ N→T› ‹Wn:n→T› ‹i∈n›
          show x(i) ∈ Wn(i) using func1_1_L1 FinProd_def ZF_fun_from_tot_val

            by simp
        qed
        ultimately have x ∈ FinProd(n→⋃T,Wn) using FinProd_def by simp
      } thus B ⊆ FinProd(n→⋃T,Wn) by auto
      next
      { fix x assume x ∈ FinProd(n→⋃T,Wn)
        then have x ∈ n→⋃T and ∀i∈domain(Wn). x(i) ∈ Wn(i)
          using  FinProd_def by auto
        with ‹Wn:n→T› and ‹N ∈ FinPow(n)› have ∀i∈N. x(i) ∈ Wn(i)
          using func1_1_L1 FinPow_def by auto
        moreover from ‹Wn:n→T› and ‹N ∈ FinPow(n)›
        have ∀i∈N. Wn(i) = W(i)
          using ZF_fun_from_tot_val FinPow_def by auto
        ultimately have ∀i∈N. x(i) ∈ W(i) by simp
        with ‹W ∈ N→T› ‹x ∈ n→⋃T› ‹B = FinProd(n→⋃T,W)› have x∈B
          using func1_1_L1 FinProd_def by simp
     } thus FinProd(n→⋃T,Wn) ⊆ B by auto
  qed
    ultimately have B ∈ {FinProd(n→⋃T,W). W∈n→T} by auto
  } thus ProductTopBase(n,T) ⊆ {FinProd(n→⋃T,W). W∈n→T} by auto
qed
```

A technical lemma providing a formula for finite product on one topological space.

```
lemma single_top_prod: assumes A1: W:1→τ
  shows FinProd(1→⋃τ,W) = { {⟨0,y⟩}. y ∈ W(0)}
proof -
  have 1 = {0} by auto
  from A1 have domain(W) = {0} using func1_1_L1 by auto
  then have FinProd(1→⋃τ,W) = {x ∈ 1→⋃τ. x(0) ∈ W(0)}
    using FinProd_def by simp
  also have {x ∈ 1→⋃τ. x(0) ∈ W(0)} = { {⟨0,y⟩}. y ∈ W(0)}
  proof
    from ‹1 = {0}› show {x ∈ 1→⋃τ. x(0) ∈ W(0)} ⊆ { {⟨0,y⟩}. y ∈ W(0)}
      using func_singleton_pair by auto
    { fix x assume x ∈ { {⟨0,y⟩}. y ∈ W(0)}
      then obtain y where x = {⟨0,y⟩} and II: y ∈ W(0) by auto
      with A1 have y ∈ ⋃τ using apply_funtype by auto
      with ‹x = {⟨0,y⟩}›  ‹1 = {0}› have x:1→⋃τ using pair_func_singleton
        by auto
      with ‹x = {⟨0,y⟩}› II have x ∈ {x ∈ 1→⋃τ. x(0) ∈ W(0)}
        using pair_val by simp
    } thus { {⟨0,y⟩}. y ∈ W(0)} ⊆ {x ∈ 1→⋃τ. x(0) ∈ W(0)} by auto
  qed
  finally show thesis by simp
qed
```

Intuitively, the topological space of singleton lists valued in X is the same as X. However, each element of this space is a list of length one, i.e a set consisting of a pair $\langle 0, x\rangle$ where x is an element of X. The next lemma provides a formula for the product topology in the corner case when we have only one factor and shows that the product topology of one space is essentially the same as the space.

```
lemma singleton_prod_top: assumes A1: τ {is a topology}
  shows
    SeqProductTopology(1,τ) = { { {⟨0,y⟩}. y∈U }. U∈τ} and
    IsAhomeomorphism(τ,SeqProductTopology(1,τ),{⟨y,{⟨0,y⟩}⟩.y ∈ ⋃τ})
proof -
  have {0} = 1 by auto
  let b = {⟨y,{⟨0,y⟩}⟩.y ∈ ⋃τ}
  have b ∈ bij(⋃τ,1→⋃τ) using list_singleton_bij by blast
  with A1 have {b(U). U∈τ} {is a topology} and IsAhomeomorphism(τ, {b(U).
U∈τ},b)
    using bij_induced_top by auto
  moreover have ∀U∈τ. b(U) = { {⟨0,y⟩}. y∈U }
  proof
    fix U assume U∈τ
    from ‹b ∈ bij(⋃τ,1→⋃τ)› have b:⋃τ→(1→⋃τ) using bij_def inj_def
      by simp
    { fix y assume y ∈ ⋃τ
      with ‹b:⋃τ→(1→⋃τ)› have b(y) = {⟨0,y⟩} using ZF_fun_from_tot_val
```

```
        by simp
    } hence ∀y ∈ ⋃τ. b(y) = {⟨0,y⟩} by auto
    with ‹U∈τ› ‹b:⋃τ→(1→⋃τ)› show  b(U) = { {⟨0,y⟩}. y∈U }
      using func_imagedef by auto
  qed
  moreover have ProductTopBase(1,τ) = { { {⟨0,y⟩}. y∈U }. U∈τ}
  proof
    { fix V assume V ∈ ProductTopBase(1,τ)
      with A1 obtain W where W:1→τ and V = FinProd(1→⋃τ,W)
        using fin_prod_def_nat by auto
      then have V ∈ { { {⟨0,y⟩}. y∈U }. U∈τ} using apply_funtype single_top_prod
        by auto
    } thus ProductTopBase(1,τ) ⊆ { { {⟨0,y⟩}. y∈U }. U∈τ} by auto
  { fix V assume V ∈ { { {⟨0,y⟩}. y∈U }. U∈τ}
    then obtain U where U∈τ and V = { {⟨0,y⟩}. y∈U } by auto
    let W = {⟨0,U⟩}
    from ‹U∈τ› have W:{0}→τ using pair_func_singleton by simp
    with ‹{0} = 1› have W:1→τ and W(0) = U using pair_val by auto
    with ‹V = { {⟨0,y⟩}. y∈U }› have V = FinProd(1→⋃τ,W)
      using single_top_prod by simp
    with A1 ‹W:1→τ› have V ∈ ProductTopBase(1,τ) using fin_prod_def_nat
      by auto
   } thus { { {⟨0,y⟩}. y∈U }. U∈τ} ⊆ ProductTopBase(1,τ) by auto
  qed
  ultimately have I: ProductTopBase(1,τ) {is a topology} and
    II: IsAhomeomorphism(τ, ProductTopBase(1,τ),b) by auto
  from A1 have ProductTopBase(1,τ) {is a base for} SeqProductTopology(1,τ)

    using seq_prod_top_is_top by simp
  with I have ProductTopBase(1,τ) = SeqProductTopology(1,τ) by (rule
base_topology)
  with ‹ProductTopBase(1,τ) = { { {⟨0,y⟩}. y∈U }. U∈τ}› II show
    SeqProductTopology(1,τ) = { { {⟨0,y⟩}. y∈U }. U∈τ} and
    IsAhomeomorphism(τ,SeqProductTopology(1,τ),{⟨y,{⟨0,y⟩}⟩.y ∈ ⋃τ}) by
auto
qed
```

A special corner case of `finite_top_prod_homeo`: a space X is homeomorphic to the space of one element lists of X.

```
theorem singleton_prod_top1: assumes A1: τ {is a topology}
  shows IsAhomeomorphism(SeqProductTopology(1,τ),τ,{⟨x,x(0)⟩. x∈1→⋃τ})
proof -
  have {⟨x,x(0)⟩. x∈1→⋃τ} = converse({⟨y,{⟨0,y⟩}⟩.y∈⋃τ})
    using list_singleton_bij by blast
  with A1 show thesis using singleton_prod_top homeo_inv by simp
qed
```

A technical lemma describing the carrier of a (cartesian) product topology of the (sequence) product topology of n copies of topology τ and another copy of τ.

```
lemma finite_prod_top: assumes τ {is a topology} and  T = SeqProductTopology(n,τ)
  shows (⋃ProductTopology(T,τ)) = (n→⋃τ)×⋃τ
  using assms Top_1_4_T1 seq_prod_top_is_top by simp
```

If U is a set from the base of X^n and V is open in X, then $U \times V$ is in the base of X^{n+1}. The next lemma is an analogue of this fact for the function space approach.

```
lemma finite_prod_succ_base: assumes A1: τ {is a topology} and A2:
n ∈ nat and
  A3: U ∈ ProductTopBase(n,τ) and A4: V∈τ
  shows {x ∈ succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V} ∈ ProductTopBase(succ(n),τ)
  proof -
    let B = {x ∈ succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V}
    from A1 A2 have ProductTopBase(n,τ) = {FinProd(n→⋃τ,W). W∈n→τ}
      using fin_prod_def_nat by simp
    with A3 obtain W_U where W_U:n→τ and U =FinProd(n→⋃τ,W_U) by auto
    let W = Append(W_U,V)
    from A4 and ‹W_U:n→τ› have W:succ(n)→τ using append_props by simp
    moreover have B = FinProd(succ(n)→⋃τ,W)
    proof
      { fix x assume x∈B
        with ‹W:succ(n)→τ› have x ∈ succ(n)→⋃τ and domain(W) = succ(n)
using func1_1_L1
          by auto
        moreover from A2 A4 ‹x∈B› ‹U =FinProd(n→⋃τ,W_U)› ‹W_U:n→τ›
‹x ∈ succ(n)→⋃τ›
        have ∀i∈succ(n). x(i) ∈ W(i) using func1_1_L1 FinProd_def init_props
append_props
          by simp
        ultimately have x ∈ FinProd(succ(n)→⋃τ,W) using FinProd_def
by simp
      } thus B ⊆ FinProd(succ(n)→⋃τ,W) by auto
      next
      { fix x assume x ∈ FinProd(succ(n)→⋃τ,W)
        then have x:succ(n)→⋃τ and I: ∀i ∈ domain(W). x(i) ∈ W(i)
          using FinProd_def by auto
        moreover have Init(x) ∈ U
        proof -
          from A2 and ‹x:succ(n)→⋃τ› have Init(x):n→⋃τ using init_props
by simp
          moreover have ∀i∈domain(W_U). Init(x)(i) ∈ W_U(i)
          proof -
            from A2 ‹x ∈ FinProd(succ(n)→⋃τ,W)› ‹W:succ(n)→τ› have
∀i∈n. x(i) ∈ W(i)
              using FinProd_def func1_1_L1 by simp
            moreover from A2 ‹x: succ(n)→⋃τ› have ∀i∈n. Init(x)(i)
= x(i)
              using init_props by simp
```

```
        moreover from A4 and ‹WU:n→τ› have ∀i∈n. W(i) =  WU(i)
          using append_props by simp
        ultimately have ∀i∈n. Init(x)(i) ∈ WU(i) by simp
        with ‹WU:n→τ› show thesis using func1_1_L1 by simp
      qed
      ultimately have Init(x) ∈ FinProd(n→⋃τ,WU) using FinProd_def
by simp
      with ‹U =FinProd(n→⋃τ,WU)› show thesis by simp
    qed
    moreover have x(n) ∈ V
    proof -
      from ‹W:succ(n)→τ› I  have x(n) ∈ W(n) using func1_1_L1 by
simp
      moreover from A4 ‹WU:n→τ› have W(n) = V using append_props
by simp
      ultimately show thesis by simp
    qed
    ultimately have x∈B by simp
  } thus FinProd(succ(n)→⋃τ,W) ⊆ B by auto
 qed
 moreover from A1 A2 have
   ProductTopBase(succ(n),τ) = {FinProd(succ(n)→⋃τ,W). W∈succ(n)→τ}
   using fin_prod_def_nat by simp
 ultimately show thesis by auto
qed
```

If U is open in X^n and V is open in X, then $U \times V$ is open in X^{n+1}. The next lemma is an analogue of this fact for the function space approach.

```
lemma finite_prod_succ: assumes A1: τ {is a topology} and A2: n ∈ nat
and
  A3: U ∈ SeqProductTopology(n,τ) and A4: V∈τ
  shows {x ∈ succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V} ∈ SeqProductTopology(succ(n),τ)
  proof -
    from A1 have ProductTopBase(n,τ) {is a base for} SeqProductTopology(n,τ)
and
      I: ProductTopBase(succ(n),τ) {is a base for} SeqProductTopology(succ(n),τ)
and
      II: SeqProductTopology(succ(n),τ) {is a topology}
        using seq_prod_top_is_top by auto
    with A3 have ∃ℬ ∈ Pow(ProductTopBase(n,τ)). U = ⋃ℬ using Top_1_2_L1
by simp
    then obtain ℬ where ℬ ⊆ ProductTopBase(n,τ) and U = ⋃ℬ by auto
    then have
    {x:succ(n)→⋃τ. Init(x) ∈ U ∧ x(n) ∈ V} = (⋃B∈ℬ.{x:succ(n)→⋃τ.
Init(x) ∈ B ∧ x(n) ∈ V})
      by auto
    moreover from A1 A2 A4 ‹ℬ ⊆ ProductTopBase(n,τ)› have
      ∀B∈ℬ. ({x:succ(n)→⋃τ. Init(x) ∈ B ∧ x(n) ∈ V} ∈ ProductTopBase(succ(n),τ))
      using finite_prod_succ_base by auto
```

```
    with I II have
      (⋃B∈ℬ.{x:succ(n)→⋃τ. Init(x) ∈ B ∧ x(n) ∈ V}) ∈ SeqProductTopology(succ(n),τ)
      using base_sets_open union_indexed_open by auto
    ultimately show thesis by simp
  qed
```

In the Topology_ZF_2 theory we define product topology of two topological spaces. The next lemma explains in what sense the topology on finite lists of length n of elements of topological space X can be thought as a model of the product topology on the cartesian product of n copies of that space. Namely, we show that the space of lists of length $n+1$ of elements of X is homeomorphic to the product topology (as defined in Topology_ZF_2) of two spaces: the space of lists of length n and X. Recall that if $\mathcal{B}$ is a base (i.e. satisfies the base condition), then the collection $\{\bigcup B | B \in Pow(\mathcal{B})\}$ is a topology (generated by $\mathcal{B}$).

```
theorem finite_top_prod_homeo: assumes A1: τ {is a topology} and A2:
n ∈ nat and
  A3: f = {⟨x,⟨Init(x),x(n)⟩⟩. x ∈ succ(n)→⋃τ} and
  A4: T = SeqProductTopology(n,τ) and
  A5: S = SeqProductTopology(succ(n),τ)
shows IsAhomeomorphism(S,ProductTopology(T,τ),f)
proof -
  let C = ProductCollection(T,τ)
  let B = ProductTopBase(succ(n),τ)
  from A1 A4 have T {is a topology} using seq_prod_top_is_top by simp
  with A1 A5  have S {is a topology} and  ProductTopology(T,τ) {is a
topology}
        using seq_prod_top_is_top  Top_1_4_T1 by auto
  moreover
  from assms have f ∈ bij(⋃S,⋃ProductTopology(T,τ))
    using lists_cart_prod seq_prod_top_is_top Top_1_4_T1 by simp
  then have f: ⋃S→⋃ProductTopology(T,τ) using bij_is_fun by simp
  ultimately have two_top_spaces0(S,ProductTopology(T,τ),f) using two_top_spaces0_def
by simp
  moreover note ‹f ∈ bij(⋃S,⋃ProductTopology(T,τ))›
  moreover from A1 A5 have B {is a base for} S
    using seq_prod_top_is_top by simp
  moreover from A1 ‹T {is a topology}› have C {is a base for} ProductTopology(T,τ)

    using Top_1_4_T1 by auto
  moreover have  ∀W∈C. f-(W) ∈ S
  proof
      fix W assume W∈C
      then obtain U V where U∈T V∈τ and W = U×V using ProductCollection_def
by auto
      from A1 A5 ‹f: ⋃S→⋃ProductTopology(T,τ)› have f: (succ(n)→⋃τ)→⋃ProductTopology
        using seq_prod_top_is_top by simp
      with assms ‹W = U×V› ‹U∈T› ‹V∈τ› show f-(W) ∈ S
```

```
        using ZF_fun_from_tot_val func1_1_L15 finite_prod_succ by simp

  qed
  moreover have ∀V∈B. f(V) ∈ ProductTopology(T,τ)
  proof
    fix V assume V∈B
    with A1 A2 obtain W_V where W_V ∈ succ(n)→τ and V = FinProd(succ(n)→⋃τ,W_V)

      using fin_prod_def_nat by auto
    let U = FinProd(n→⋃τ,Init(W_V))
    let W = W_V(n)
    have U ∈ T
    proof -
      from A1 A2 ‹W_V ∈ succ(n)→τ› have U ∈ ProductTopBase(n,τ)
        using fin_prod_def_nat init_props by auto
      with A1 A4 show thesis using seq_prod_top_is_top base_sets_open
by blast
    qed
    from A1 ‹W_V ∈ succ(n)→τ› ‹T {is a topology}› ‹U ∈ T› have U×W ∈
ProductTopology(T,τ)
      using apply_funtype prod_open_open_prod by simp
    moreover have f(V) = U×W
    proof -
      from A2 ‹W_V: succ(n)→τ› have Init(W_V): n→τ and III: ∀k∈n. Init(W_V)(k)
= W_V(k)
        using init_props by auto
      then have domain(Init(W_V)) = n using func1_1_L1 by simp
      have f(V) = {⟨Init(x),x(n)⟩. x∈V}
      proof -
        have f(V) = {f(x). x∈V}
        proof -
          from A1 A5 have B {is a base for} S using seq_prod_top_is_top
by simp
          with ‹V∈B› have V ⊆ ⋃S using IsAbaseFor_def by auto
          with ‹f: ⋃S→⋃ProductTopology(T,τ)› show thesis using func_imagedef
by simp
        qed
        moreover have ∀x∈V. f(x) = ⟨Init(x),x(n)⟩
        proof -
          from A1 A3 A5 ‹V = FinProd(succ(n)→⋃τ,W_V)› have V ⊆ ⋃S and

            fdef: f = {⟨x,⟨Init(x),x(n)⟩⟩. x ∈ ⋃S} using seq_prod_top_is_top
FinProd_def
            by auto
          from ‹f: ⋃S→⋃ProductTopology(T,τ)› fdef have ∀x ∈ ⋃S. f(x)
= ⟨Init(x),x(n)⟩
            by (rule ZF_fun_from_tot_val0)
          with ‹V ⊆ ⋃S› show thesis by auto
        qed
```

```
      ultimately show thesis by simp
    qed
    also have {⟨Init(x),x(n)⟩. x∈V} = U×W
    proof
      { fix y assume y ∈ {⟨Init(x),x(n)⟩. x∈V}
        then obtain x where I: y = ⟨Init(x),x(n)⟩ and x∈V by auto

        with ‹V = FinProd(succ(n)→⋃τ,W_V)› have
          x:succ(n)→⋃τ and II: ∀k∈domain(W_V). x(k) ∈ W_V(k)
          unfolding FinProd_def by auto
        with A2 ‹W_V: succ(n)→τ› have IV: ∀k∈n. Init(x)(k) = x(k)
          using init_props by simp
        have Init(x) ∈ U
        proof -
          from A2 ‹x:succ(n)→⋃τ› have Init(x): n→⋃τ using init_props
by simp
          moreover have ∀k∈domain(Init(W_V)). Init(x)(k) ∈ Init(W_V)(k)
          proof -
            from A2 ‹W_V: succ(n)→τ› have Init(W_V): n→τ using init_props
by simp
            then have domain(Init(W_V)) = n using func1_1_L1 by simp
            note III IV  ‹domain(Init(W_V)) = n›
            moreover from II ‹W_V ∈ succ(n)→τ› have ∀k∈n. x(k) ∈
W_V(k)
              using func1_1_L1 by simp
            ultimately show thesis by simp
          qed
          ultimately show Init(x) ∈ U using FinProd_def by simp
        qed
        moreover from ‹W_V: succ(n)→τ› II have x(n) ∈ W using func1_1_L1
by simp
        ultimately have ⟨Init(x),x(n)⟩ ∈ U×W by simp
        with I have y ∈ U×W by simp
      } thus {⟨Init(x),x(n)⟩. x∈V} ⊆ U×W by auto
      { fix y assume y ∈ U×W
        then have fst(y) ∈ U and snd(y) ∈ W by auto
        with ‹domain(Init(W_V)) = n› have fst(y): n→⋃τ and
          V: ∀k∈n. fst(y)(k) ∈ Init(W_V)(k)
          using FinProd_def by auto
        from ‹W_V: succ(n)→τ› have W ∈ τ using apply_funtype by simp
        with ‹snd(y) ∈ W› have snd(y) ∈ ⋃τ by auto
        let x = Append(fst(y),snd(y))
        have x∈V
        proof -
          from ‹fst(y): n→⋃τ› ‹snd(y) ∈ ⋃τ› have x:succ(n)→⋃τ
using append_props by simp
          moreover have ∀i∈domain(W_V). x(i) ∈ W_V(i)
          proof -
            from ‹fst(y): n→⋃τ› ‹snd(y) ∈ ⋃τ›
```

```
              have ∀k∈n. x(k) = fst(y)(k) and x(n) = snd(y)
              using append_props by auto
            moreover from III V have ∀k∈n. fst(y)(k) ∈ W_V(k) by simp

            moreover note ‹snd(y) ∈ W›
            ultimately have ∀i∈succ(n). x(i) ∈ W_V(i) by simp
            with ‹W_V ∈ succ(n)→τ› show thesis using func1_1_L1 by
simp
          qed
          ultimately have x ∈ FinProd(succ(n)→⋃τ,W_V) using FinProd_def
by simp
          with ‹V = FinProd(succ(n)→⋃τ,W_V)› show x∈V by simp
        qed
        moreover from A2 ‹y ∈ U×W› ‹fst(y): n→⋃τ› ‹snd(y) ∈ ⋃τ›
have y = ⟨Init(x),x(n)⟩
          using init_append append_props by auto
        ultimately have y ∈ {⟨Init(x),x(n)⟩. x∈V} by auto
      } thus U×W ⊆ {⟨Init(x),x(n)⟩. x∈V} by auto
    qed
    finally show f(V) = U×W by simp
  qed
  ultimately show f(V) ∈ ProductTopology(T,τ) by simp
 qed
 ultimately show thesis using two_top_spaces0.bij_base_open_homeo by
simp
qed


end
```

78 Topology - examples

```
theory Topology_ZF_examples imports Topology_ZF Cardinal_ZF

begin
```

This theory deals with some concrete examples of topologies.

78.1 CoCardinal Topology

In this section we define and prove the basic properties of the co-cardinal topology on a set X.

The collection of subsets of a set whose complement is strictly bounded by a cardinal is a topology given some assumptions on the cardinal.

```
definition
CoCardinal(X,T) ≡ {F∈Pow(X). X-F ≺ T}∪ {0}
```

For any set and any infinite cardinal we prove that `CoCardinal(X,Q)` forms a topology. The proof is done with an infinite cardinal, but it is obvious that the set `Q` can be any set equipollent with an infinite cardinal. It is a topology also if the set where the topology is defined is too small or the cardinal too large; in this case, as it is later proved the topology is a discrete topology. And the last case corresponds with `Q=1` which translates in the indiscrete topology.

```
lemma CoCar_is_topology:
  assumes InfCard (Q)
  shows CoCardinal(X,Q) {is a topology}
proof -
  let T = CoCardinal(X,Q)
  {
    fix M
    assume A:M∈Pow(T)
    hence M⊆T by auto
    then have M⊆Pow(X) using CoCardinal_def by auto
    then have ⋃M∈Pow(X) by auto
    moreover
    {
      assume B:M=0
      then have ⋃M∈T using CoCardinal_def by auto
    }
    moreover
    {
      assume B:M={0}
      then have ⋃M∈T using CoCardinal_def by auto
    }
    moreover
    {
      assume B:M ≠0 M≠{0}
      from B obtain T where C:T∈M and T≠0 by auto
      with A have D:X-T ≺ (Q) using CoCardinal_def by auto
      from C have X-⋃M⊆X-T by blast
      with D have X-⋃M≺ (Q) using subset_imp_lepoll lesspoll_trans1
by blast
    }
    ultimately have ⋃M∈T using CoCardinal_def by auto
  }
  moreover
  {
    fix U and V
    assume U∈T and V∈T
    then have A:U=0 ∨ (U∈Pow(X) ∧ X-U≺ (Q)) and
      B:V=0 ∨ (V∈Pow(X) ∧ X-V≺ (Q)) using CoCardinal_def by auto
    hence D:U∈Pow(X)V∈Pow(X) by auto
    have C:X-(U ∩ V)=(X-U)∪(X-V) by fast
    with A B C have U∩V=0∨(U∩V∈Pow(X) ∧ X-(U ∩ V)≺ (Q)) using less_less_imp_un_less
```

```
assms
      by auto
    then have U∩V∈T using CoCardinal_def by auto
  }
  ultimately show thesis using IsATopology_def by auto
qed
```

We can use theorems proven in topology0 context for the co-cardinal topology.

```
theorem topology0_CoCardinal:
  assumes InfCard(T)
  shows topology0(CoCardinal(X,T))
  using topology0_def CoCar_is_topology assms by auto
```

It can also be proven that if CoCardinal(X,T) is a topology, X≠0, Card(T) and T≠0; then T is an infinite cardinal, X≺T or T=1. It follows from the fact that the union of two closed sets is closed. Choosing the appropriate cardinals, the cofinite and the cocountable topologies are obtained.

The cofinite topology is a very special topology because it is closely related to the separation axiom T_1. It also appears naturally in algebraic geometry.

```
definition
  Cofinite (CoFinite _ 90) where
  CoFinite X ≡ CoCardinal(X,nat)
```

Cocountable topology in fact consists of the empty set and all cocountable subsets of X.

```
definition
  Cocountable (CoCountable _ 90) where
  CoCountable X ≡ CoCardinal(X,csucc(nat))
```

78.2 Total set, Closed sets, Interior, Closure and Boundary

There are several assertions that can be done to the CoCardinal(X,T) topology. In each case, we will not assume sufficient conditions for CoCardinal(X,T) to be a topology, but they will be enough to do the calculations in every posible case.

The topology is defined in the set X

```
lemma union_cocardinal:
  assumes T≠0
  shows ⋃ CoCardinal(X,T) = X
proof-
  have X:X-X=0 by auto
  have 0 ≲ 0 by auto
  with assms have 0≺11 ≲T using not_0_is_lepoll_1 lepoll_imp_lesspoll_succ
by auto
  then have 0≺T using lesspoll_trans2  by auto
```

```
  with X have (X-X)≺T by auto
  then have X∈CoCardinal(X,T) using CoCardinal_def by auto
  hence X⊆⋃ CoCardinal(X,T) by blast
  then show ⋃ CoCardinal(X,T)=X using CoCardinal_def by auto
qed
```

The closed sets are the small subsets of X and X itself.

```
lemma closed_sets_cocardinal:
  assumes T≠0
  shows D {is closed in} CoCardinal(X,T) ⟷ (D∈Pow(X) ∧ D≺T) ∨ D=X
proof-
  {
    assume A:D ⊆ X X - D ∈ CoCardinal(X,T)  D ≠ X
    from A(1,3) have X-(X-D)=D X-D≠0 by auto
    with A(2) have D≺T using CoCardinal_def by simp
  }
  with assms have D {is closed in} CoCardinal(X,T) ⟶ (D∈Pow(X) ∧  D≺T)∨
D=X using IsClosed_def
    union_cocardinal by auto
  moreover
  {
    assume A:D ≺ TD ⊆ X
    from A(2) have X-(X-D)=D by blast
    with A(1) have X-(X-D)≺ T by auto
    then have X-D∈ CoCardinal(X,T) using CoCardinal_def by auto
  }
  with assms have (D∈Pow(X) ∧  D≺T)⟶ D {is closed in} CoCardinal(X,T)
using union_cocardinal
    IsClosed_def by auto
  moreover
  have X-X=0 by auto
  then have X-X∈ CoCardinal(X,T)using CoCardinal_def by auto
  with assms have X{is closed in} CoCardinal(X,T) using union_cocardinal
    IsClosed_def by auto
  ultimately show thesis by auto
qed
```

The interior of a set is itself if it is open or `0` if it isn't open.

```
lemma interior_set_cocardinal:
  assumes noC: T≠0 and A⊆X
  shows Interior(A,CoCardinal(X,T))= (if ((X-A) ≺ T) then A else 0)
proof-
  from assms(2) have dif_dif:X-(X-A)=A by blast
  {
    assume (X-A) ≺ T
    then have (X-A)∈Pow(X) ∧ (X-A) ≺ T by auto
    with noC have (X-A) {is closed in} CoCardinal(X,T) using closed_sets_cocardinal
      by auto
    with noC have X-(X-A)∈CoCardinal(X,T) using IsClosed_def union_cocardinal
```

```
      by auto
    with dif_dif have A∈CoCardinal(X,T) by auto
    hence A∈{U∈CoCardinal(X,T). U ⊆ A} by auto
    hence a1:A⊆⋃{U∈CoCardinal(X,T). U ⊆ A} by auto
    have a2:⋃{U∈CoCardinal(X,T). U ⊆ A}⊆A by blast
    from a1 a2 have Interior(A,CoCardinal(X,T))=A using Interior_def
by auto}
  moreover
  {
    assume as:~((X-A) ≺ T)
    {
      fix U
      assume U ⊆A
      hence X-A ⊆ X-U by blast
      then have Q:X-A ≲ X-U using subset_imp_lepoll by auto
      {
        assume X-U≺ T
        with Q have X-A≺ T using lesspoll_trans1 by auto
        with as have False  by auto
      }
      hence ~((X-U) ≺ T) by auto
      then have U∉CoCardinal(X,T)∨U=0 using CoCardinal_def by auto
    }
    hence {U∈CoCardinal(X,T). U ⊆ A}⊆{0} by blast
    then have Interior(A,CoCardinal(X,T))=0 using Interior_def by auto
  }
  ultimately show thesis by auto
qed
```

X is a closed set that contains A. This lemma is necessary because we cannot use the lemmas proven in the topology0 context since $T \neq 0$ is too weak for CoCardinal(X,T) to be a topology.

```
lemma X_closedcov_cocardinal:
  assumes T≠0 A⊆X
  shows X ∈ ClosedCovers(A,CoCardinal(X,T)) using ClosedCovers_def
  using union_cocardinal closed_sets_cocardinal assms by auto
```

The closure of a set is itself if it is closed or X if it isn't closed.

```
lemma closure_set_cocardinal:
  assumes T≠0A⊆X
  shows Closure(A,CoCardinal(X,T))=(if (A ≺ T) then A else X)
proof-
  {
    assume A ≺ T
    with assms have A {is closed in} CoCardinal(X,T) using closed_sets_cocardinal
by auto
    with assms(2) have A∈ {D ∈ Pow(X). D {is closed in} CoCardinal(X,T)
∧ A⊆D} by auto
    with assms(1) have S:A∈ClosedCovers(A,CoCardinal(X,T)) using ClosedCovers_def
```

```
      using union_cocardinal by auto
    hence l1:⋂ClosedCovers(A,CoCardinal(X,T))⊆A by blast
    from S have l2:A ⊆ ⋂ClosedCovers(A,CoCardinal(X,T))
      unfolding  ClosedCovers_def by auto
    from l1 l2 have Closure(A,CoCardinal(X,T))=A using Closure_def
      by auto
  }
  moreover
  {
    assume as:¬ A ≺ T
    {
      fix U
      assume A⊆U
      then have Q:A ≲ U using subset_imp_lepoll by auto
      {
        assume U≺ T
        with Q have A≺ T using lesspoll_trans1 by auto
        with as have False by auto
      }
      hence ¬ U ≺ T by auto
      with assms(1) have ¬(U {is closed in} CoCardinal(X,T)) ∨ U=X us-
ing closed_sets_cocardinal
      by auto
    }
    with assms(1) have ∀U∈Pow(X). U{is closed in}CoCardinal(X,T) ∧ A⊆U⟶U=X
      by auto
    with assms(1) have ClosedCovers(A,CoCardinal(X,T))⊆{X}
      using union_cocardinal using ClosedCovers_def by auto
    with assms have ClosedCovers(A,CoCardinal(X,T))={X} using X_closedcov_cocardinal
      by auto
    then have  Closure(A,CoCardinal(X,T)) = X  using Closure_def by auto
  }
  ultimately show thesis by auto
qed
```

The boundary of a set is empty if A and $X - A$ are closed, `X` if not A neither $X - A$ are closed and; if only one is closed, then the closed one is its boundary.

```
lemma boundary_cocardinal:
  assumes T≠0A⊆X
  shows Boundary(A,CoCardinal(X,T)) = (if A≺ T then (if  (X-A)≺ T then
0 else A) else (if  (X-A)≺ T then X-A else X))
proof-
  from assms(2) have X-A ⊆ X by auto
  {
    assume AS: A≺T X-A ≺ T
    with assms ‹X-A ⊆ X› have
      Closure(X-A,CoCardinal(X,T)) = X-A and Closure(A,CoCardinal(X,T))
= A
```

```
      using closure_set_cocardinal by auto
    with assms(1) have Boundary(A,CoCardinal(X,T)) = 0
      using Boundary_def  union_cocardinal by auto
  }
  moreover
  {
    assume AS: ~(A≺T) X-A ≺ T
    with assms ‹X-A ⊆ X› have
      Closure(X-A,CoCardinal(X,T)) = X-A and Closure(A,CoCardinal(X,T))
= X
      using closure_set_cocardinal by auto
    with assms(1) have Boundary(A,CoCardinal(X,T))=X-A using Boundary_def
      union_cocardinal by auto
  }
  moreover
  {
    assume AS:~(A≺T) ~(X-A ≺ T)
    with assms ‹X-A ⊆ X› have
      Closure(X-A,CoCardinal(X,T))=X and Closure(A,CoCardinal(X,T))=X
      using closure_set_cocardinal by auto
    with assms(1) have Boundary(A,CoCardinal(X,T))=X using Boundary_def
union_cocardinal
      by auto
  }
  moreover
  {
    assume AS:A≺ T ~(X-A≺T)
    with assms ‹X-A ⊆ X› have
      Closure(X-A,CoCardinal(X,T))=X and Closure(A,CoCardinal(X,T)) =
A
      using closure_set_cocardinal by auto
    with assms have Boundary(A,CoCardinal(X,T))=A using Boundary_def
union_cocardinal
      by auto
  }
  ultimately show thesis by auto
qed
```

If the set is too small or the cardinal too large, then the topology is just the discrete topology.

```
lemma discrete_cocardinal:
  assumes X≺T
  shows CoCardinal(X,T) = Pow(X)
proof
  {
    fix U
    assume U∈CoCardinal(X,T)
    then have U ∈ Pow(X) using CoCardinal_def by auto
  }
```

```
  then show CoCardinal(X,T) ⊆ Pow(X) by auto
  {
    fix U
    assume A:U ∈ Pow(X)
    then have X-U ⊆ X by auto
    then have X-U ≲X using subset_imp_lepoll by auto
    then have X-U≺ T using lesspoll_trans1 assms by auto
    with A have U∈CoCardinal(X,T) using CoCardinal_def
      by auto
  }
  then show Pow(X) ⊆ CoCardinal(X,T) by auto
qed
```

If the cardinal is taken as T=1 then the topology is indiscrete.

```
lemma indiscrete_cocardinal:
  shows CoCardinal(X,1) = {0,X}
proof
  {
    fix Q
    assume Q ∈ CoCardinal(X,1)
    then have Q ∈ Pow(X) and Q=0 ∨ X-Q≺1 using CoCardinal_def by auto
    then have Q ∈ Pow(X) and Q=0 ∨ X-Q=0 using lesspoll_succ_iff lepoll_0_iff
by auto
    then have Q=0 ∨ Q=X by blast
  }
  then show CoCardinal(X,1) ⊆ {0, X} by auto
  have 0 ∈ CoCardinal(X,1) using CoCardinal_def by auto
  moreover
  have 0≺1 and X-X=0 using lesspoll_succ_iff by auto
  then have X∈CoCardinal(X,1) using CoCardinal_def by auto
  ultimately show {0, X} ⊆ CoCardinal(X,1)  by auto
qed
```

The topological subspaces of the CoCardinal(X,T) topology are also CoCardinal topologies.

```
lemma subspace_cocardinal:
  shows CoCardinal(X,T) {restricted to} Y = CoCardinal(Y∩X,T)
proof
  {
    fix M
    assume M ∈ (CoCardinal(X,T) {restricted to} Y)
    then obtain A where A1:A ∈ CoCardinal(X,T) M=Y ∩ A using RestrictedTo_def
by auto
    then have M ∈ Pow(X ∩ Y) using CoCardinal_def by auto
    moreover
    from A1 have (Y ∩ X)-M = (Y ∩ X)-A using CoCardinal_def by auto
    with ‹(Y ∩ X)-M = (Y ∩ X)-A› have (Y ∩ X)-M⊆ X-A by auto
    then have (Y ∩ X)-M ≲ X-A using subset_imp_lepoll by auto
    with A1 have (Y ∩ X)-M ≺ T ∨ M=0 using lesspoll_trans1 CoCardinal_def
```

```
      by auto
    ultimately have M ∈ CoCardinal(Y∩X, T) using CoCardinal_def
      by auto
  }
  then show CoCardinal(X,T) {restricted to} Y ⊆ CoCardinal(Y∩X,T) by
auto
  {
    fix M
    let A = M ∪ (X-Y)
    assume A:M ∈ CoCardinal(Y ∩ X,T)
    {
      assume M=0
      hence M=0 ∩ Y by auto
      then have M∈CoCardinal(X,T) {restricted to} Y using RestrictedTo_def
        CoCardinal_def by auto
    }
    moreover
    {
      assume AS:M≠0
      from A AS have A1:(M∈Pow(Y ∩ X) ∧ (Y ∩ X)-M≺T) using CoCardinal_def
by auto
      hence A∈Pow(X) by blast
      moreover
      have X-A=(Y ∩ X)-M by blast
      with A1 have X-A≺ T by auto
      ultimately have A∈CoCardinal(X,T) using CoCardinal_def by auto
      then have AT:Y ∩ A∈CoCardinal(X,T) {restricted to} Y using RestrictedTo_def
        by auto
      have Y ∩ A=Y ∩ M by blast
      also from A1 have ...=M by auto
      finally have Y ∩ A=M by simp
      with AT have M∈CoCardinal(X,T) {restricted to} Y
        by auto
    }
    ultimately have M∈CoCardinal(X,T) {restricted to} Y by auto
  }
  then show CoCardinal(Y ∩ X, T) ⊆ CoCardinal(X,T) {restricted to} Y
by auto
qed
```

78.3 Excluded Set Topology

In this section, we consider all the subsets of a set which have empty intersection with a fixed set.

The excluded set topology consists of subsets of X that are disjoint with a fixed set U.

```
definition ExcludedSet(X,U) ≡ {F∈Pow(X). U ∩ F=0}∪ {X}
```

For any set; we prove that ExcludedSet(X,Q) forms a topology.

```
theorem excludedset_is_topology:
  shows ExcludedSet(X,Q) {is a topology}
proof-
  {
    fix M
    assume M ∈ Pow(ExcludedSet(X,Q))
    then have A:M⊆{F∈Pow(X). Q ∩ F=0}∪ {X} using ExcludedSet_def by
auto
    hence ⋃M∈Pow(X) by auto
    moreover
    {
      have B:Q ∩⋃M=⋃{Q ∩T. T∈M} by auto
      {
        assume X∉M
        with A have M⊆{F∈Pow(X). Q ∩ F=0} by auto
        with B have Q ∩ ⋃M=0 by auto
      }
      moreover
      {
        assume X∈M
        with A have ⋃M=X by auto
      }
      ultimately have  Q ∩ ⋃M=0 ∨ ⋃M=X by auto
    }
    ultimately have ⋃M∈ExcludedSet(X,Q) using ExcludedSet_def by auto
  }
  moreover
  {
    fix U V
    assume U∈ExcludedSet(X,Q) V∈ExcludedSet(X,Q)
    then have U∈Pow(X)V∈Pow(X)U=X∨ U ∩ Q=0V=X∨ V ∩ Q=0 using ExcludedSet_def
by auto
    hence U∈Pow(X)V∈Pow(X)(U ∩ V)=X ∨ Q∩(U ∩ V)=0 by auto
    then have (U ∩ V)∈ExcludedSet(X,Q) using ExcludedSet_def by auto
  }
  ultimately show thesis using IsATopology_def by auto
qed
```

We can use topology0 when discussing excluded set topology.

```
theorem topology0_excludedset:
  shows topology0(ExcludedSet(X,T))
  using topology0_def excludedset_is_topology by auto
```

Choosing a singleton set, it is considered a point in excluded topology.

```
definition
  ExcludedPoint(X,p) ≡ ExcludedSet(X,{p})
```

78.4 Total set, closed sets, interior, closure and boundary

Here we discuss what are closed sets, interior, closure and boundary in excluded set topology.

The topology is defined in the set X

```
lemma union_excludedset:
  shows ⋃ExcludedSet(X,T) = X
proof-
  have X ∈ExcludedSet(X,T) using ExcludedSet_def by auto
  then show thesis using ExcludedSet_def by auto
qed
```

The closed sets are those which contain the set (X ∩ T) and 0.

```
lemma closed_sets_excludedset:
  shows D {is closed in}ExcludedSet(X,T) ⟷ (D∈Pow(X) ∧ (X ∩ T) ⊆ D)
∨ D=0
proof-
  {
    fix x
    assume A:D ⊆ X X-D ∈ExcludedSet(X,T) D≠0 x∈T x∈X
    from A(1) have B:X-(X-D)=D by auto
    from A(2) have T∩(X-D)=0∨ X-D=X using ExcludedSet_def by auto
    hence T∩(X-D)=0∨ X-(X-D)=X-X by auto
    with B have T∩(X-D)=0∨ D=X-X by auto
    hence T∩(X-D)=0∨ D=0 by auto
    with A(3) have T∩(X-D)=0 by auto
    with A(4) have x∉X-D by auto
    with A(5) have x∈D by auto
  }
  moreover
  {
    assume A:X∩T⊆D D⊆X
    from A(1) have X-D⊆X-(X∩T) by auto
    also have ... = X-T by auto
    finally have T∩(X-D) = 0 by auto
    moreover
    have X-D ∈ Pow(X) by auto
    ultimately have X-D ∈ExcludedSet(X,T) using ExcludedSet_def by auto
  }
  ultimately show thesis using IsClosed_def union_excludedset ExcludedSet_def

    by auto
qed
```

The interior of a set is itself if it is X or the difference with the set T

```
lemma interior_set_excludedset:
  assumes A⊆X
  shows Interior(A,ExcludedSet(X,T)) = (if A=X then X else A-T)
```

```
proof-
  {
    assume A:A≠X
    from assms have A-T ∈ExcludedSet(X,T) using ExcludedSet_def by auto
    then have A-T⊆Interior(A,ExcludedSet(X,T))
    using Interior_def by auto
    moreover
    {
      fix U
      assume U ∈ExcludedSet(X,T) U⊆A
      then have T∩U=0 ∨ U=XU⊆A using ExcludedSet_def by auto
      with A assms have T∩U=0U⊆A by auto
      then have U-T=UU-T⊆A-T by auto
      then have U⊆A-T by auto
    }
    then have Interior(A,ExcludedSet(X,T))⊆A-T using Interior_def by
auto
    ultimately have Interior(A,ExcludedSet(X,T))=A-T by auto
  }
  moreover
  have X∈ExcludedSet(X,T) using ExcludedSet_def
  union_excludedset by auto
  then have Interior(X,ExcludedSet(X,T)) = X using topology0.Top_2_L3
  topology0_excludedset by auto
  ultimately show thesis by auto
qed
```

The closure of a set is itself if it is 0 or the union with T.

```
lemma closure_set_excludedset:
  assumes A⊆X
  shows Closure(A,ExcludedSet(X,T))=(if A=0 then 0 else A ∪(X∩ T))
proof-
  have 0∈ClosedCovers(0,ExcludedSet(X,T)) using ClosedCovers_def
    closed_sets_excludedset by auto
  then have Closure(0,ExcludedSet(X,T))⊆0 using Closure_def by auto
  hence Closure(0,ExcludedSet(X,T))=0 by blast
  moreover
  {
    assume A:A≠0
    with assms have (A∪(X∩T)) {is closed in}ExcludedSet(X,T) using closed_sets_excludedset

      by blast
    then have (A ∪(X∩ T))∈ {D ∈ Pow(X). D {is closed in}ExcludedSet(X,T)
∧ A⊆D}
    using assms by auto
    then have (A ∪(X∩ T))∈ClosedCovers(A,ExcludedSet(X,T)) unfolding
ClosedCovers_def
    using union_excludedset by auto
    then have l1:⋂ClosedCovers(A,ExcludedSet(X,T)) ⊆ (A ∪(X∩ T)) by
```

```
blast
    {
      fix U
      assume U∈ClosedCovers(A,ExcludedSet(X,T))
      then have U{is closed in}ExcludedSet(X,T) and A⊆U using ClosedCovers_def
       union_excludedset by auto
      then have U=0∨(X∩T)⊆U and A⊆U using closed_sets_excludedset
       by auto
      with A have (X∩T)⊆UA⊆U by auto
      hence (X∩T)∪A⊆U by auto
    }
    with assms have (A ∪(X∩ T)) ⊆ ⋂ClosedCovers(A,ExcludedSet(X,T))

      using topology0.Top_3_L3 topology0_excludedset union_excludedset

      by auto
    with l1 have ⋂ClosedCovers(A,ExcludedSet(X,T)) = (A∪(X∩T)) by auto
    then have Closure(A, ExcludedSet(X,T)) = A∪(X∩T) using Closure_def

      by auto
  }
  ultimately show thesis by auto
qed
```

The boundary of a set is `0` if A is `X` or `0`, and `X∩T` in other case.

```
lemma boundary_excludedset:
  assumes A⊆X
  shows Boundary(A,ExcludedSet(X,T)) = (if A=0∨A=X then 0 else X∩T)
proof-
  {
    have Closure(0,ExcludedSet(X,T))=0Closure(X - 0,ExcludedSet(X,T))=X
    using closure_set_excludedset by auto
    then have Boundary(0,ExcludedSet(X,T)) = 0using Boundary_def using
      union_excludedset assms by auto
  }
  moreover
  {
    have X-X=0 by blast
    then have Closure(X,ExcludedSet(X,T)) = X and Closure(X-X,ExcludedSet(X,T))
= 0
    using closure_set_excludedset by auto
    then have Boundary(X,ExcludedSet(X,T)) = 0unfolding Boundary_def us-
ing
      union_excludedset by auto
  }
  moreover
  {
    assume A≠0 and A≠X
    then have X-A≠0 using assms by auto
```

```
    with assms ‹A≠0› ‹A⊆X› have Closure(A,ExcludedSet(X,T)) = A ∪ (X∩T)
      using closure_set_excludedset by simp
    moreover
    from ‹A⊆X› have X-A ⊆ X by blast
    with ‹X-A≠0› have Closure(X-A,ExcludedSet(X,T)) = (X-A) ∪ (X∩T)
      using closure_set_excludedset by simp
    ultimately have Boundary(A,ExcludedSet(X,T)) = X∩T
      using Boundary_def union_excludedset by auto
  }
  ultimately show thesis by auto
qed
```

78.5 Special cases and subspaces

This section provides some miscellaneous facts about excluded set topologies.

The excluded set topology is equal in the sets `T` and `X∩T`.

```
lemma smaller_excludedset:
  shows ExcludedSet(X,T) = ExcludedSet(X,(X∩T))
proof
  show ExcludedSet(X,T) ⊆ ExcludedSet(X, X∩T) and ExcludedSet(X, X∩T)
⊆ExcludedSet(X,T)
    unfolding ExcludedSet_def by auto
qed
```

If the set which is excluded is disjoint with `X`, then the topology is discrete.

```
lemma empty_excludedset:
  assumes T∩X=0
  shows ExcludedSet(X,T) = Pow(X)
proof
  from assms show ExcludedSet(X,T) ⊆ Pow(X)  using smaller_excludedset
ExcludedSet_def
    by auto
  from assms show Pow(X) ⊆ExcludedSet(X,T) unfolding ExcludedSet_def
by blast
qed
```

The topological subspaces of the `ExcludedSet X T` topology are also ExcludedSet topologies.

```
lemma subspace_excludedset:
  shows ExcludedSet(X,T) {restricted to} Y = ExcludedSet(Y ∩ X, T)
proof
  {
    fix M
    assume M∈(ExcludedSet(X,T) {restricted to} Y)
    then obtain A where A1:A:ExcludedSet(X,T) M=Y ∩ A unfolding RestrictedTo_def
by auto
```

```
    then have M∈Pow(X ∩ Y) unfolding ExcludedSet_def by auto
    moreover
    from A1 have T∩M=0∨M=Y∩X unfolding ExcludedSet_def by blast
    ultimately have M ∈ ExcludedSet(Y ∩ X,T) unfolding ExcludedSet_def
      by auto
  }
  then show ExcludedSet(X,T) {restricted to} Y ⊆ ExcludedSet(Y∩X,T)
by auto
  {
    fix M
    let A = M  ∪ ((X∩Y-T)-Y)
    assume A:M ∈ ExcludedSet(Y∩X,T)
    {
      assume M = Y ∩ X
      then have M ∈ ExcludedSet(X,T) {restricted to} Y unfolding RestrictedTo_def
        ExcludedSet_def by auto
    }
    moreover
    {
      assume AS:M≠Y ∩ X
      from A AS have A1:(M∈Pow(Y ∩ X) ∧ T∩M=0) unfolding ExcludedSet_def
by auto
      then have A∈Pow(X) by blast
      moreover
      have T∩A=T∩M by blast
      with A1 have T∩A=0 by auto
      ultimately have A ∈ExcludedSet(X,T) unfolding ExcludedSet_def by
auto
      then have AT:Y ∩ A ∈ExcludedSet(X,T) {restricted to} Y unfold-
ing RestrictedTo_def
        by auto
      have Y ∩ A=Y ∩ M by blast
      also have ...=M using A1 by auto
      finally have Y∩A = M by simp
      with AT have M ∈ExcludedSet(X,T) {restricted to} Y by auto
    }
    ultimately have M ∈ExcludedSet(X,T) {restricted to} Y by auto
  }
  then show ExcludedSet(Y ∩ X,T) ⊆ ExcludedSet(X,T) {restricted to}
Y by auto
qed
```

78.6 Included Set Topology

In this section we consider the subsets of a set which contain a fixed set. The family defined in this section and the one in the previous section are dual; meaning that the closed set of one are the open sets of the other.

We define the included set topology as the collection of supersets of some fixed subset of the space X.

```
definition
  IncludedSet(X,U) ≡ {F∈Pow(X). U ⊆ F} ∪ {0}
```

In the next theorem we prove that `IncludedSet X Q` forms a topology.

```
theorem includedset_is_topology:
  shows IncludedSet(X,Q) {is a topology}
proof-
  {
    fix M
    assume M ∈ Pow(IncludedSet(X,Q))
    then have A:M⊆{F∈Pow(X). Q ⊆ F}∪ {0} using IncludedSet_def by auto
    then have ⋃M∈Pow(X) by auto
    moreover
    haveQ ⊆⋃M∨ ⋃M=0 using A by blast
    ultimately have ⋃M∈IncludedSet(X,Q) using IncludedSet_def by auto
  }
  moreover
  {
    fix U V
    assume U∈IncludedSet(X,Q) V∈IncludedSet(X,Q)
    then have U∈Pow(X)V∈Pow(X)U=0∨ Q⊆UV=0∨ Q⊆V using IncludedSet_def
by auto
    then have U∈Pow(X)V∈Pow(X)(U ∩ V)=0 ∨ Q⊆(U ∩ V) by auto
    then have (U ∩ V)∈IncludedSet(X,Q) using IncludedSet_def by auto
  }
  ultimately show thesis using IsATopology_def by auto
qed
```

We can reference the theorems proven in the `topology0` context when discussing the included set topology.

```
theorem topology0_includedset:
  shows topology0(IncludedSet(X,T))
  using topology0_def includedset_is_topology by auto
```

Choosing a singleton set, it is considered a point excluded topology. In the following lemmas and theorems, when neccessary it will be considered that `T≠0` and `T⊆X`. These cases will appear in the special cases section.

```
definition
  IncludedPoint (IncludedPoint _ _ 90) where
  IncludedPoint X p ≡ IncludedSet(X,{p})
```

78.7 Basic topological notions in included set topology

This section discusses total set, closed sets, interior, closure and boundary for included set topology.

The topology is defined in the set X.

```
lemma union_includedset:
  assumes T⊆X
  shows ⋃IncludedSet(X,T) = X
proof-
  from assms have X ∈ IncludedSet(X,T) using IncludedSet_def by auto
  then show ⋃IncludedSet(X,T) = X using IncludedSet_def by auto
qed
```

The closed sets are those which are disjoint with T and X.

```
lemma closed_sets_includedset:
  assumes T⊆X
  shows D {is closed in} IncludedSet(X,T) ⟷ (D∈Pow(X) ∧  (D ∩ T)=0)∨
D=X
proof-
  have X-X=0 by blast
  then have X-X∈IncludedSet(X,T) using IncludedSet_def by auto
  moreover
  {
    assume A:D ⊆ X X - D ∈ IncludedSet(X,T)  D ≠ X
    from A(2) have T⊆(X-D)∨ X-D=0 using IncludedSet_def by auto
    with A(1) have T⊆(X-D)∨ D=X by blast
    with A(3) have T⊆(X-D) by auto
    hence D∩T=0 by blast
  }
  moreover
  {
    assume A:D∩T=0D⊆X
    from A(1) assms have T⊆(X-D) by blast
    then have X-D∈IncludedSet(X,T) using IncludedSet_def by auto
  }
  ultimately show thesis using IsClosed_def union_includedset assms by
auto
qed
```

The interior of a set is itself if it is open or the empty set if it isn't.

```
lemma interior_set_includedset:
  assumes A⊆X
  shows Interior(A,IncludedSet(X,T))= (if T⊆A then A else 0)
proof-
  {
    fix x
    assume A:Interior(A,IncludedSet(X,T)) ≠ 0 x∈T
    have Interior(A,IncludedSet(X,T)) ∈ IncludedSet(X,T) using
      topology0.Top_2_L2 topology0_includedset by auto
    with A(1) have T ⊆ Interior(A,IncludedSet(X,T)) using IncludedSet_def
      by auto
    with A(2) have x ∈ Interior(A,IncludedSet(X,T)) by auto
    then have x∈A using topology0.Top_2_L1 topology0_includedset by auto}
    moreover
```

```
  {
    assume T⊆A
    with assms have A∈IncludedSet(X,T) using IncludedSet_def by auto
    then have Interior(A,IncludedSet(X,T)) = A using topology0.Top_2_L3
      topology0_includedset by auto
  }
  ultimately show thesis by auto
qed
```

The closure of a set is itself if it is closed or the whole space if it is not.

```
lemma closure_set_includedset:
  assumes A⊆X T⊆X
  shows Closure(A,IncludedSet(X,T)) = (if T∩A=0 then A else X)
proof-
  {
    assume AS:T∩A=0
    then have A {is closed in} IncludedSet(X,T) using closed_sets_includedset
      assms by auto
    with assms(1) have Closure(A,IncludedSet(X,T))=A using topology0.Top_3_L8
      topology0_includedset union_includedset assms(2) by auto
  }
  moreover
  {
    assume AS:T∩A ≠ 0
    have X∈ClosedCovers(A,IncludedSet(X,T)) using ClosedCovers_def
      closed_sets_includedset union_includedset assms by auto
    then have l1:⋂ClosedCovers(A,IncludedSet(X,T))⊆X using Closure_def
      by auto
    moreover
    {
      fix U
      assume U∈ClosedCovers(A,IncludedSet(X,T))
      then have U{is closed in}IncludedSet(X,T)A⊆U using ClosedCovers_def
        by auto
      then have U=X∨(T∩U)=0A⊆U using closed_sets_includedset assms(2)
        by auto
      then have U=X∨(T∩A)=0 by auto
      then have U=X using AS by auto
    }
    then have X ⊆ ⋂ClosedCovers(A,IncludedSet(X,T)) using topology0.Top_3_L3
      topology0_includedset union_includedset assms by auto
    ultimately have ⋂ClosedCovers(A,IncludedSet(X,T))=X by auto
    then have Closure(A,IncludedSet(X,T)) = X
      using Closure_def by auto
  }
  ultimately show thesis by auto
qed
```

The boundary of a set is X-A if A contains T completely, is A if $X-A$ contains

T completely and X if T is divided between the two sets. The case where T=0 is considered as a special case.

```
lemma boundary_includedset:
  assumes A⊆X T⊆X T≠0
  shows Boundary(A,IncludedSet(X,T))=(if T⊆A then X-A else (if T∩A=0
then A else X))
proof -
  from ‹A⊆X› have X-A ⊆ X by auto
  {
    assume T⊆A
    with assms(2,3) have T∩A≠0 and T∩(X-A)=0 by auto
    with assms(1,2)  ‹X-A ⊆ X›  have
      Closure(A,IncludedSet(X,T)) = X and Closure(X-A,IncludedSet(X,T))
= (X-A)
      using closure_set_includedset by auto
    with assms(2) have Boundary(A,IncludedSet(X,T)) = X-A
      using Boundary_def union_includedset by auto
  }
  moreover
  {
    assume ~(T⊆A) and T∩A=0
    with assms(2) have T∩(X-A)≠0 by auto
    with assms(1,2) ‹T∩A=0› ‹X-A ⊆ X› have
      Closure(A,IncludedSet(X,T)) = A and Closure(X-A,IncludedSet(X,T))
= X
      using  closure_set_includedset by auto
    with assms(1,2) have Boundary(A,IncludedSet(X,T))=A using Boundary_def
union_includedset
      by auto
  }
  moreover
  {
    assume ~(T⊆A) and T∩A ≠ 0
    with assms(1,2) have T∩(X-A) ≠ 0 by auto
    with assms(1,2) ‹T∩A≠0›  ‹X-A ⊆ X› have
      Closure(A,IncludedSet(X,T)) = X and Closure(X-A,IncludedSet(X,T))
= X
      using closure_set_includedset by auto
    with assms(2) have Boundary(A,IncludedSet(X,T)) = X
      using Boundary_def union_includedset  by auto
  }
  ultimately show thesis by auto
qed
```

78.8 Special cases and subspaces

In this section we discuss some corner cases when some parameters in our definitions are empty and provide some facts about subspaces in included set topologies.

The topology is discrete if `T=0`

```
lemma smaller_includedset:
  shows IncludedSet(X,0) = Pow(X)
proof
  show IncludedSet(X,0) ⊆ Pow(X) and Pow(X) ⊆ IncludedSet(X,0)
    unfolding IncludedSet_def by auto
qed
```

If the set which is included is not a subset of `X`, then the topology is trivial.

```
lemma empty_includedset:
  assumes ~(T⊆X)
  shows IncludedSet(X,T) = {0}
proof
  from assms show IncludedSet(X,T) ⊆ {0} and {0} ⊆ IncludedSet(X,T)
    unfolding IncludedSet_def by auto
qed
```

The topological subspaces of the `IncludedSet(X,T)` topology are also IncludedSet topologies. The trivial case does not fit the idea in the demonstration because if `Y⊆X` then `IncludedSet(Y∩X, Y∩T)` is never trivial. There is no need for a separate proof because the only subspace of the trivial topology is itself.

```
lemma subspace_includedset:
  assumes T⊆X
  shows IncludedSet(X,T) {restricted to} Y = IncludedSet(Y∩X,Y∩T)
proof
  {
    fix M
    assume M ∈ (IncludedSet(X,T) {restricted to} Y)
    then obtain A where A1:A:IncludedSet(X,T) M = Y∩A unfolding RestrictedTo_def

      by auto
    then have M ∈ Pow(X∩Y) unfolding IncludedSet_def by auto
    moreover
    from A1 have Y∩T⊆M ∨ M=0 unfolding IncludedSet_def by blast
    ultimately have M ∈ IncludedSet(Y∩X, Y∩T) unfolding IncludedSet_def
      by auto
  }
  then show IncludedSet(X,T) {restricted to} Y ⊆ IncludedSet(Y∩X, Y∩T)

    by auto
  {
    fix M
    let A = M ∪ T
    assume A:M ∈ IncludedSet(Y∩X, Y∩T)
    {
```

```
      assume M=0
      then have M∈IncludedSet(X,T) {restricted to} Y unfolding RestrictedTo_def
        IncludedSet_def by auto
    }
    moreover
    {
      assume AS:M≠0
      from A AS have A1:M∈Pow(Y∩X) ∧ Y∩T⊆M unfolding IncludedSet_def
by auto
      then have A∈Pow(X) using assms by blast
      moreover
      have T⊆A by blast
      ultimately have A ∈ IncludedSet(X,T) unfolding IncludedSet_def by
auto
      then have AT:Y ∩ A ∈ IncludedSet(X,T) {restricted to} Yunfolding
RestrictedTo_def
        by auto
      from A1 have Y ∩ A=Y ∩ M by blast
      also from A1 have ...=M by auto
      finally have Y∩A = M by simp
      with AT have M ∈ IncludedSet(X,T) {restricted to} Y
        by auto
    }
    ultimately have M ∈ IncludedSet(X,T) {restricted to} Y by auto
  }
  thus IncludedSet(Y∩X, Y∩T) ⊆ IncludedSet(X,T) {restricted to} Y by
auto
qed

end
```

79 More examples in topology

```
theory Topology_ZF_examples_1
imports Topology_ZF_1 Order_ZF
begin
```

In this theory file we reformulate the concepts related to a topology in relation with a base of the topology and we give examples of topologies defined by bases or subbases.

79.1 New ideas using a base for a topology

79.2 The topology of a base

Given a family of subsets satisfying the base condition, it is possible to construct a topology where that family is a base of. Even more, it is the only topology with such characteristics.

```
definition
  TopologyWithBase (TopologyBase _  50) where
  U {satisfies the base condition} ⟹ TopologyBase U ≡ THE T. U {is a
base for} T
```

If a collection U of sets satisfies the base condition then the topology constructed from it is indeed a topology and U is a base for this topology.

```
theorem Base_topology_is_a_topology:
  assumes U {satisfies the base condition}
  shows (TopologyBase U) {is a topology} and U {is a base for} (TopologyBase
U)
proof-
  from assms obtain T where U {is a base for} T using
    Top_1_2_T1(2) by blast
  then have ∃!T. U {is a base for} T using same_base_same_top ex1I[where
P=
    λT. U {is a base for} T] by blast
  with assms show U {is a base for} (TopologyBase U)  using theI[where
P=
    λT. U {is a base for} T] TopologyWithBase_def by auto
  with assms  show (TopologyBase U) {is a topology} using Top_1_2_T1(1)
    IsAbaseFor_def by auto
qed
```

A base doesn't need the empty set.

```
lemma base_no_0:
  shows B{is a base for}T ⟷ (B-{0}){is a base for}T
proof-
  {
    fix M
    assume M∈{⋃A . A ∈ Pow(B)}
    then obtain Q where M=⋃QQ∈Pow(B) by auto
    hence M=⋃(Q-{0})Q-{0}∈Pow(B-{0}) by auto
    hence M∈{⋃A . A ∈ Pow(B - {0})} by auto
  }
  hence {⋃A . A ∈ Pow(B)} ⊆ {⋃A . A ∈ Pow(B - {0})} by blast
  moreover
  {
    fix M
    assume M∈{⋃A . A ∈ Pow(B-{0})}
    then obtain Q where M=⋃QQ∈Pow(B-{0}) by auto
    hence M=⋃(Q)Q∈Pow(B) by auto
    hence M∈{⋃A . A ∈ Pow(B)} by auto
  }
  hence  {⋃A . A ∈ Pow(B - {0})} ⊆ {⋃A . A ∈ Pow(B)}
    by auto
  ultimately have {⋃A . A ∈ Pow(B - {0})} = {⋃A . A ∈ Pow(B)}  by auto
  then show B{is a base for}T ⟷ (B-{0}){is a base for}T using IsAbaseFor_def
by auto
```

qed

The interior of a set is the union of all the sets of the base which are fully contained by it.

```
lemma interior_set_base_topology:
  assumes U {is a base for} T T{is a topology}
  shows Interior(A,T) = ⋃{T∈U. T⊆A}
proof
  have {T∈U. T⊆A}⊆U by auto
  with assms(1) have ⋃{T∈U. T⊆A}∈T
    using IsAbaseFor_def by auto
  moreover have ⋃{T∈U. T⊆A} ⊆ A by blast
  ultimately show  ⋃{T∈U. T⊆A} ⊆ Interior(A,T)
    using assms(2) topology0.Top_2_L5 topology0_def by auto
  {
    fix x
    assume x ∈ Interior(A,T)
    with assms obtain V where V∈U V ⊆ Interior(A,T) x∈V
      using point_open_base_neigh topology0.Top_2_L2 topology0_def
      by blast
    with assms have V∈U x∈V V⊆A using topology0.Top_2_L1 topology0_def
      by auto
    hence x ∈ ⋃{T∈U. T⊆A} by auto
  }
  thus Interior(A, T) ⊆ ⋃{T ∈ U . T ⊆ A}  by auto
qed
```

In the following, we offer another lemma about the closure of a set given a basis for a topology. This lemma is based on `cl_inter_neigh` and `inter_neigh_cl`. It states that it is only necessary to check the sets of the base, not all the open sets.

```
lemma closure_set_base_topology:
  assumes U {is a base for} Q Q{is a topology} A⊆⋃Q
  shows Closure(A,Q) = {x∈⋃Q. ∀T∈U. x∈T⟶A∩T≠0}
proof
  {
    fix x
    assume A:x∈Closure(A,Q)
    with assms(2,3) have B:x∈⋃Q using topology0_def topology0.Top_3_L11(1)
      by blast
    moreover
    {
      fix T
      assume T∈U x∈T
      with assms(1) have T∈Qx∈T using base_sets_open by auto
      with assms(2,3) A have A∩T ≠ 0 using topology0_def topology0.cl_inter_neigh
        by auto
    }
    hence ∀T∈U. x∈T⟶A∩T≠0 by auto
```

```
    ultimately have x∈{x∈⋃Q. ∀T∈U. x∈T⟶A∩T≠0} by auto
  }
  thus Closure(A, Q) ⊆{x∈⋃Q. ∀T∈U. x∈T⟶A∩T≠0}
    by auto
  {
    fix x
    assume AS:x∈{x ∈ ⋃Q . ∀T∈U. x ∈ T ⟶ A ∩ T ≠ 0}
    hence x∈⋃Q by blast
    moreover
    {
      fix R
      assume R∈Q
      with assms(1) obtain W where RR:W⊆U R=⋃W using
        IsAbaseFor_def by auto
      {
        assume x∈R
        with RR(2) obtain WW where TT:WW∈Wx∈WW by auto
        {
          assume R∩A=0
          with RR(2) TT(1) have WW∩A=0  by auto
           with TT(1) RR(1) have WW∈U WW∩A=0 by auto
          with AS have x∈⋃Q-WW by auto
          with TT(2) have False by auto
        }
        hence R∩A≠0 by auto
      }
    }
    hence ∀U∈Q. x∈U ⟶ U∩A≠0 by auto
    ultimately have x∈Closure(A,Q) using assms(2,3) topology0_def topology0.inter_neigh_cl

      by auto
  }
  then show {x ∈ ⋃Q . ∀T∈U. x ∈ T ⟶ A ∩ T ≠ 0} ⊆ Closure(A,Q)
    by auto
qed
```

The restriction of a base is a base for the restriction.

```
lemma subspace_base_topology:
  assumes B {is a base for} T
  shows (B {restricted to} Y) {is a base for} (T {restricted to} Y)
proof -
  from assms have (B {restricted to} Y) ⊆ (T {restricted to} Y)
    unfolding IsAbaseFor_def RestrictedTo_def by auto
  moreover have (T {restricted to} Y) = {⋃A. A ∈ Pow(B {restricted to}
Y)}
  proof
    { fix U assume U ∈ (T {restricted to} Y)
      then obtain W where W∈T and U = W∩Y unfolding RestrictedTo_def
by blast
```

```
      with assms obtain C where C∈Pow(B) and W=⋃C unfolding IsAbaseFor_def
        by blast
      let A={V∩Y. V∈C}
      from ‹C∈Pow(B)› ‹U = W∩Y› ‹W=⋃C› have
        A ∈ Pow(B {restricted to} Y) and U=(⋃A)
        unfolding RestrictedTo_def by auto
      hence U ∈ {⋃A. A ∈ Pow(B {restricted to} Y)} by blast
    } thus (T {restricted to} Y) ⊆ {⋃A. A ∈ Pow(B {restricted to} Y)}
      by auto
    { fix U assume U ∈ {⋃A. A ∈ Pow(B {restricted to} Y)}
      then obtain A where A: A ⊆ (B {restricted to} Y) and U = (⋃A)
by auto
      let A0={C∈B. Y∩C∈A}
      from A have A0 ⊆ B and A = A0 {restricted to} Y unfolding RestrictedTo_def
        by auto
      with ‹U = (⋃A)› have A0 ⊆ B and U = ⋃(A0 {restricted to} Y)
        by auto
      with assms have U ∈ (T {restricted to} Y) unfolding RestrictedTo_def
IsAbaseFor_def
        by auto
    } thus {⋃A. A ∈ Pow(B {restricted to} Y)} ⊆ (T {restricted to} Y)
by blast
  qed
  ultimately show thesis unfolding IsAbaseFor_def by simp
qed
```

If the base of a topology is contained in the base of another topology, then the topologies maintain the same relation.

```
theorem base_subset:
  assumes B{is a base for}TB2{is a base for}T2B⊆B2
  shows T⊆T2
proof
  {
    fix x
    assume x∈T
    with assms(1) obtain M where M⊆Bx=⋃M using IsAbaseFor_def by auto
    with assms(3) have M⊆B2x=⋃M by auto
    with assms(2) show x∈T2 using IsAbaseFor_def by auto
  }
qed
```

79.3 Dual Base for Closed Sets

A dual base for closed sets is the collection of complements of sets of a base for the topology.

```
definition
  DualBase (DualBase _ _ 80) where
  B{is a base for}T ⟹ DualBase B T≡{⋃T-U. U∈B}∪{⋃T}
```

```
lemma closed_inter_dual_base:
  assumes D{is closed in}TB{is a base for}T
  obtains M where M⊆DualBase B TD=⋂M
proof-
  assume K:⋀M. M ⊆ DualBase B T ⟹ D = ⋂M ⟹ thesis
  {
    assume AS:D≠⋃T
    from assms(1) have D:D∈Pow(⋃T)⋃T-D∈T using IsClosed_def by auto
    hence A:⋃T-(⋃T-D)=D⋃T-D∈T by auto
    with assms(2) obtain Q where QQ:Q∈Pow(B)⋃T-D=⋃Q using IsAbaseFor_def
by auto
    {
      assume Q=0
      then have ⋃Q=0 by auto
      with QQ(2) have ⋃T-D=0 by auto
      with D(1) have D=⋃T by auto
      with AS have False by auto
    }
    hence QNN:Q≠0 by auto
    from QQ(2) A(1) have D=⋃T-⋃Q by auto
    with QNN have D=⋂{⋃T-R. R∈Q} by auto
    moreover
    with assms(2) QQ(1) have {⋃T-R. R∈Q}⊆DualBase B T using DualBase_def
      by auto
    with calculation K have thesis by auto
  }
  moreover
  {
    assume AS:D=⋃T
    with assms(2) have {⋃T}⊆DualBase B T using DualBase_def by auto
    moreover
    have ⋃T = ⋂{⋃T} by auto
    with calculation K AS have thesis by auto
  }
  ultimately show thesis by auto
qed
```

We have already seen for a base that whenever there is a union of open sets, we can consider only basic open sets due to the fact that any open set is a union of basic open sets. What we should expect now is that when there is an intersection of closed sets, we can consider only dual basic closed sets.

```
lemma closure_dual_base:
  assumes U {is a base for} QQ{is a topology}A⊆⋃Q
  shows Closure(A,Q)=⋂{T∈DualBase U Q. A⊆T}
proof
  from assms(1) have T:⋃Q∈DualBase U Q using DualBase_def by auto
  moreover
```

```
  {
    fix T
    assume A:T∈DualBase U Q A⊆T
    with assms(1) obtain R where (T=⋃Q-R∧R∈U)∨T=⋃Q using DualBase_def
      by auto
    with A(2) assms(1,2) have  (T{is closed in}Q)A⊆TT∈Pow(⋃Q) using
topology0.Top_3_L1 topology0_def
      topology0.Top_3_L9 base_sets_open by auto
  }
  then have SUB:{T∈DualBase U Q. A⊆T}⊆{T∈Pow(⋃Q). (T{is closed in}Q)∧A⊆T}
    by blast
  with calculation assms(3) have ⋂{T∈Pow(⋃Q). (T{is closed in}Q)∧A⊆T}⊆⋂{T∈DualBase
U Q. A⊆T}
    by auto
  then show Closure(A,Q)⊆⋂{T∈DualBase U Q. A⊆T} using Closure_def ClosedCovers_def
    by auto
  {
    fix x
    assume A:x∈⋂{T∈DualBase U Q. A⊆T}
    {
      fix T
      assume B:x∈TT∈U
      {
        assume C:A∩T=0
        from B(2) assms(1) have ⋃Q-T∈DualBase U Q using DualBase_def
          by auto
        moreover
        from C assms(3) have A⊆⋃Q-T by auto
        moreover
        from B(1) have x∉⋃Q-T by auto
        ultimately have x∉⋂{T∈DualBase U Q. A⊆T} by auto
        with A have False by auto
      }
      hence A∩T≠0 by auto
    }
    hence ∀T∈U. x∈T⟶A∩T≠0 by auto
    moreover
    from T A assms(3) have x∈⋃Q by auto
    with calculation assms have x∈Closure(A,Q) using closure_set_base_topology
     by auto
  }
  thus ⋂{T ∈ DualBase U Q . A ⊆ T} ⊆ Closure(A, Q) by auto
qed
```

79.4 Partition topology

In the theory file Partitions_ZF.thy; there is a definition to work with partitions. In this setting is much easier to work with a family of subsets.

definition

```
  IsAPartition (_{is a partition of}_ 90) where
  (U {is a partition of} X) ≡ (⋃U=X ∧ (∀A∈U. ∀B∈U. A=B∨ A∩B=0)∧ 0∉U)
```

A subcollection of a partition is a partition of its union.

```
lemma subpartition:
  assumes U {is a partition of} X V⊆U
  shows V{is a partition of}⋃V
  using assms unfolding IsAPartition_def by auto
```

A restriction of a partition is a partition. If the empty set appears it has to be removed.

```
lemma restriction_partition:
  assumes U {is a partition of}X
  shows ((U {restricted to} Y)-{0}) {is a partition of} (X∩Y)
  using assms unfolding IsAPartition_def RestrictedTo_def
  by fast
```

Given a partition, the complement of a union of a subfamily is a union of a subfamily.

```
lemma diff_union_is_union_diff:
  assumes R⊆P P {is a partition of} X
  shows X - ⋃R=⋃(P-R)
proof
  {
    fix x
    assume x∈X - ⋃R
    hence P:x∈Xx∉⋃R by auto
    {
      fix T
      assume T∈R
      with P(2) have x∉T by auto
    }
    with P(1) assms(2) obtain Q where Q∈(P-R)x∈Q using IsAPartition_def
by auto
    hence x∈⋃(P-R) by auto
  }
  thus X - ⋃R⊆⋃(P-R) by auto
  {
    fix x
    assume x∈⋃(P-R)
    then obtain Q where Q∈P-Rx∈Q by auto
    hence C: Q∈PQ∉Rx∈Q by auto
    then have x∈⋃P by auto
    with assms(2) have x∈X using IsAPartition_def by auto
    moreover
    {
      assume x∈⋃R
      then obtain t where G:t∈R x∈t by auto
```

```
      with C(3) assms(1) have t∩Q≠0t∈P by auto
      with assms(2) C(1,3) have t=Q using IsAPartition_def
        by blast
      with C(2) G(1) have False by auto
    }
    hence x∉⋃R by auto
    ultimately have x∈X-⋃R by auto
  }
  thus ⋃(P-R)⊆X - ⋃R by auto
qed
```

79.5 Partition topology is a topology.

A partition satisfies the base condition.

```
lemma partition_base_condition:
  assumes P {is a partition of} X
  shows P {satisfies the base condition}
proof-
  {
    fix U V
    assume AS:U∈P∧V∈P
    with assms have A:U=V∨ U∩V=0 using IsAPartition_def by auto
    {
      fix x
      assume ASS:x∈U∩V
      with A have U=V by auto
      with AS ASS have U∈Px∈U∧ U⊆U∩V by auto
      hence ∃W∈P. x∈W∧ W⊆U∩V by auto
    }
    hence (∀x ∈ U∩V. ∃W∈P. x∈W ∧ W ⊆ U∩V) by auto
  }
  then show thesis using SatisfiesBaseCondition_def by auto
qed
```

Since a partition is a base of a topology, and this topology is uniquely determined; we can built it. In the definition we have to make sure that we have a partition.

```
definition
  PartitionTopology (PTopology _ _ 50) where
  (U {is a partition of} X) ⟹ PTopology X U ≡ TopologyBase U

theorem Ptopology_is_a_topology:
  assumes U {is a partition of} X
  shows (PTopology X U) {is a topology} and U {is a base for} (PTopology
X U)
  using assms Base_topology_is_a_topology partition_base_condition
    PartitionTopology_def by auto
```

```
lemma topology0_ptopology:
  assumes U {is a partition of} X
  shows topology0(PTopology X U)
  using Ptopology_is_a_topology topology0_def assms by auto
```

79.6 Total set, Closed sets, Interior, Closure and Boundary

The topology is defined in the set X

```
lemma union_ptopology:
  assumes U {is a partition of} X
  shows ⋃(PTopology X U)=X
  using assms Ptopology_is_a_topology(2) Top_1_2_L5
    IsAPartition_def by auto
```

The closed sets are the open sets.

```
lemma closed_sets_ptopology:
  assumes T {is a partition of} X
  showsD {is closed in} (PTopology X T) ⟷ D∈(PTopology X T)
proof
  from assms
  have B:T{is a base for}(PTopology X T) using Ptopology_is_a_topology(2)
by auto
  {
    fix D
    assume D {is closed in} (PTopology X T)
    with assms have A:D∈Pow(X)X-D∈(PTopology X T)
      using IsClosed_def union_ptopology by auto
    from A(2) B obtain R where Q:R⊆T X-D=⋃R using Top_1_2_L1[where
B=T and U=X-D]
      by auto
    from A(1) have X-(X-D)=D by blast
    with Q(2) have D=X-⋃R by auto
    with Q(1) assms have D=⋃(T-R) using diff_union_is_union_diff
      by auto
    with B show D∈(PTopology X T) using IsAbaseFor_def by auto
  }
  {
    fix D
    assume D∈(PTopology X T)
    with B obtain R where Q:R⊆TD=⋃R using IsAbaseFor_def by auto
    hence X-D=X-⋃R by auto
    with Q(1) assms have X-D=⋃(T-R) using diff_union_is_union_diff
      by auto
    with B have X-D∈(PTopology X T) using IsAbaseFor_def by auto
    moreover
    from Q have D⊆⋃T by auto
    with assms have D⊆X using IsAPartition_def by auto
    with calculation assms show D{is closed in} (PTopology X T)
      using IsClosed_def union_ptopology by auto
```

```
  }
qed
```

There is a formula for the interior given by an intersection of sets of the dual base. Is the intersection of all the closed sets of the dual basis such that they do not complement A to X. Since the interior of X must be inside X, we have to enter X as one of the sets to be intersected.

```
lemma interior_set_ptopology:
  assumes U {is a partition of} XA⊆X
  shows Interior(A,(PTopology X U))=⋂{T∈DualBase U (PTopology X U).
T=X∨T∪A≠X}
proof
  {
    fix x
    assume x∈Interior(A,(PTopology X U))
    with assms obtain R where A:x∈RR∈(PTopology X U)R⊆A
      using topology0.open_open_neigh topology0_ptopology
      topology0.Top_2_L2 topology0.Top_2_L1
      by auto
    with assms obtain B where B:B⊆UR=⋃B using Ptopology_is_a_topology(2)
      IsAbaseFor_def by auto
    from A(1,3) assms have XX:x∈XX∈{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}
      using union_ptopology[of UX] DualBase_def[ofU] Ptopology_is_a_topology(2)[of
UX] by (safe,blast,auto)
    moreover
    from B(2) A(1) obtain S where C:S∈Bx∈S by auto
    {
      fix T
      assume AS:T∈DualBase U (PTopology X U)T ∪A≠X
      from AS(1) assms obtain w where (T=X-w∧w∈U)∨(T=X)
        using DualBase_def union_ptopology Ptopology_is_a_topology(2)
        by auto
      with assms(2) AS(2) have D:T=X-ww∈U by auto
      from D(2) have w⊆⋃U by auto
      with assms(1) have w⊆⋃(PTopology X U) using Ptopology_is_a_topology(2)
Top_1_2_L5[of UPTopology X U]
        by auto
      with assms(1) have w⊆X using union_ptopology by auto
      with D(1) have X-T=w by auto
      with D(2) have X-T∈U by auto
      {
        assume x∈X-T
        with C B(1) have S∈US∩(X-T)≠0 by auto
        with ‹X-T∈U› assms(1) have X-T=S using IsAPartition_def by auto
        with ‹X-T=w›‹T=X-w› have X-S=T by auto
        with AS(2) have X-S∪A≠X by auto
        from A(3) B(2) C(1) have S⊆A by auto
        hence X-A⊆X-S by auto
        with assms(2) have X-S∪A=X  by auto
```

```
        with ‹X-S∪A≠X› have False by auto
        }
      then have x∈T using XX by auto
      }
    ultimately have x∈⋂{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}
      by auto
  }
  thus Interior(A,(PTopology X U))⊆⋂{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}
by auto
  {
    fix x
    assume p:x∈⋂{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}
    hence noE:⋂{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}≠0 by auto
    {
      fix T
      assume T∈DualBase U (PTopology X U)
      with assms(1) obtain w where T=X∨(w∈U∧T=X-w) using DualBase_def
        Ptopology_is_a_topology(2) union_ptopology by auto
      with assms(1) have T=X∨(w∈(PTopology X U)∧T=X-w) using base_sets_open
        Ptopology_is_a_topology(2) by blast
      with assms(1) have T{is closed in}(PTopology X U) using topology0.Top_3_L1[where
T=PTopology X U]
        topology0_ptopology topology0.Top_3_L9[where T=PTopology X U]
union_ptopology
        by auto
    }
    moreover
    from assms(1) p have X∈{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}and
X:x∈X using Ptopology_is_a_topology(2)
      DualBase_def union_ptopology by auto
    with calculation assms(1) have (⋂{T∈DualBase U (PTopology X U).
T=X∨T∪A≠X}) {is closed in}(PTopology X U)
      using topology0.Top_3_L4[where K={T∈DualBase U (PTopology X U).
T=X∨T∪A≠X}] topology0_ptopology[where U=U and X=X]
      by auto
    with assms(1) have ab:(⋂{T∈DualBase U (PTopology X U). T=X∨T∪A≠X})∈(PTopology
X U)
      using closed_sets_ptopology by auto
    with assms(1) obtain B where B∈Pow(U)(⋂{T∈DualBase U (PTopology
X U). T=X∨T∪A≠X})=⋃B
      using Ptopology_is_a_topology(2) IsAbaseFor_def by auto
    with p obtain R where x∈RR∈UR⊆(⋂{T∈DualBase U (PTopology X U).
T=X∨T∪A≠X})
      by auto
    with assms(1) have R:x∈RR∈(PTopology X U)R⊆(⋂{T∈DualBase U (PTopology
X U). T=X∨T∪A≠X})X-R∈DualBase U (PTopology X U)
      using base_sets_open Ptopology_is_a_topology(2) DualBase_def union_ptopology
      by (safe,blast,simp,blast)
    {
```

```
      assume (X-R) ∪A≠X
      with R(4) have X-R∈{T∈DualBase U (PTopology X U). T=X∨T∪A≠X} by
auto
      hence ⋂{T∈DualBase U (PTopology X U). T=X∨T∪A≠X}⊆X-R by auto
      with R(3) have R⊆X-R using subset_trans[where A=R and C=X-R] by
auto
      hence R=0 by blast
      with R(1) have False by auto
    }
    hence I:(X-R) ∪A=X by auto
    {
      fix y
      assume ASR:y∈R
      with R(2) have y∈⋃(PTopology X U) by auto
      with assms(1) have XX:y∈X using union_ptopology by auto
      with I have y∈(X-R) ∪A by auto
      with XX have y∉R∨y∈A by auto
      with ASR have y∈A by auto
    }
    hence R⊆A by auto
    with R(1,2) have ∃R∈(PTopology X U). (x∈R∧R⊆A) by auto
    with assms(1) have x∈Interior(A,(PTopology X U)) using topology0.Top_2_L6
      topology0_ptopology by auto
  }
  thus ⋂{T ∈ DualBase U PTopology X U . T = X ∨ T ∪ A ≠ X} ⊆ Interior(A,
PTopology X U)
    by auto
qed
```

The closure of a set is the union of all the sets of the partition which intersect with A.

```
lemma closure_set_ptopology:
  assumes U {is a partition of} XA⊆X
  shows Closure(A,(PTopology X U))=⋃{T∈U. T∩A≠0}
proof
  {
    fix x
    assume A:x∈Closure(A,(PTopology X U))
    with assms have x∈⋃(PTopology X U) using topology0.Top_3_L11(1)[where
T=PTopology X U
      and A=A] topology0_ptopology union_ptopology by auto
    with assms(1) have x∈⋃U using Top_1_2_L5[where B=U and T=PTopology
X U] Ptopology_is_a_topology(2) by auto
    then obtain W where B:x∈WW∈U by auto
    with A have x∈Closure(A,(PTopology X U))∩W by auto
    moreover
    from assms B(2) have W∈(PTopology X U)A⊆X using base_sets_open Ptopology_is_a_topology
      by (safe,blast)
    with calculation assms(1) have A∩W≠0 using topology0_ptopology[where
```

```
U=U and X=X]
      topology0.cl_inter_neigh union_ptopology by auto
    with B have x∈⋃{T∈U. T∩A≠0} by blast
  }
  thus Closure(A, PTopology X U) ⊆ ⋃{T ∈ U . T ∩ A ≠ 0} by auto
  {
    fix x
    assume x∈⋃{T ∈ U . T ∩ A ≠ 0}
    then obtain T where A:x∈TT∈UT∩A≠0 by auto
    from assms have A⊆⋃(PTopology X U) using union_ptopology by auto
    moreover
    from A(1,2) assms(1) have x∈⋃(PTopology X U) using Top_1_2_L5[where
B=U and T=PTopology X U]
      Ptopology_is_a_topology(2) by auto
    moreover
    {
      fix Q
      assume B:Q∈(PTopology X U)x∈Q
      with assms(1) obtain M where C:Q=⋃MM⊆U using
        Ptopology_is_a_topology(2)
        IsAbaseFor_def by auto
      from B(2) C(1) obtain R where D:R∈Mx∈R by auto
      with C(2) A(1,2) have R∩T≠0R∈UT∈U by auto
      with assms(1) have R=T using IsAPartition_def by auto
      with C(1) D(1) have T⊆Q by auto
      with A(3) have Q∩A≠0 by auto
    }
    then have ∀Q∈(PTopology X U). x∈Q ⟶ Q∩A≠0 by auto
    with calculation assms(1) have x∈Closure(A,(PTopology X U)) using
topology0.inter_neigh_cl
      topology0_ptopology by auto
  }
  then show ⋃{T ∈ U . T ∩ A ≠ 0} ⊆ Closure(A, PTopology X U)  by auto
qed
```

The boundary of a set is given by the union of the sets of the partition which have non empty intersection with the set but that are not fully contained in it. Another equivalent statement would be: the union of the sets of the partition which have non empty intersection with the set and its complement.

```
lemma boundary_set_ptopology:
  assumes U {is a partition of} XA⊆X
  shows Boundary(A,(PTopology X U))=⋃{T∈U. T∩A≠0 ∧ ~(T⊆A)}
proof-
  from assms have Closure(A,(PTopology X U))=⋃{T ∈ U . T ∩ A ≠ 0} us-
ing
    closure_set_ptopology by auto
  moreover
  from assms(1) have Interior(A,(PTopology X U))=⋃{T ∈ U . T ⊆ A} us-
ing
```

```
    interior_set_base_topology Ptopology_is_a_topology[where U=U and
X=X] by auto
  with calculation assms have A:Boundary(A,(PTopology X U))=⋃{T ∈ U
. T ∩ A ≠ 0} - ⋃{T ∈ U . T ⊆ A}
    using topology0.Top_3_L12 topology0_ptopology union_ptopology
    by auto
  from assms(1) have ({T ∈ U . T ∩ A ≠ 0}) {is a partition of} ⋃({T
∈ U . T ∩ A ≠ 0})
    using subpartition by blast
  moreover
  {
    fix T
    assume T∈UT⊆A
    with assms(1) have T∩A=TT≠0 using IsAPartition_def by auto
    with ‹T∈U› have T∩A≠0T∈U by auto
  }
  then have {T ∈ U . T ⊆ A}⊆{T ∈ U . T ∩ A ≠ 0}  by auto
  ultimately have ⋃{T ∈ U . T ∩ A ≠ 0} - ⋃{T ∈ U . T ⊆ A}=⋃(({T ∈
U . T ∩ A ≠ 0})-({T ∈ U . T ⊆ A}))
  using diff_union_is_union_diff by auto
  also have ...=⋃({T ∈ U . T ∩ A ≠ 0 ∧ ~(T⊆A)}) by blast
  with calculation A show thesis by auto
qed
```

79.7 Special cases and subspaces

The discrete and the indiscrete topologies appear as special cases of this partition topologies.

```
lemma discrete_partition:
  shows {{x}.x∈X} {is a partition of}X
  using IsAPartition_def by auto

lemma indiscrete_partition:
  assumes X≠0
  shows {X} {is a partition of} X
  using assms IsAPartition_def by auto

theorem discrete_ptopology:
  shows (PTopology X {{x}.x∈X})=Pow(X)
proof
  {
    fix t
    assume t∈(PTopology X {{x}.x∈X})
    hence t⊆⋃(PTopology X {{x}.x∈X}) by auto
    then have t∈Pow(X) using union_ptopology
      discrete_partition by auto
  }
  thus (PTopology X {{x}.x∈X})⊆Pow(X) by auto
  {
```

```
    fix t
    assume A:t∈Pow(X)
    have ⋃({{x}. x∈t})=t by auto
    moreover
    from A have {{x}. x∈t}∈Pow({{x}.x∈X}) by auto
    hence ⋃({{x}. x∈t})∈{⋃A . A ∈ Pow({{x} . x ∈ X})} by auto
    ultimately
    have t∈(PTopology X {{x} . x ∈ X}) using Ptopology_is_a_topology(2)
      discrete_partition IsAbaseFor_def by auto
  }
  thus Pow(X) ⊆ (PTopology X {{x} . x ∈ X})  by auto
qed

theorem indiscrete_ptopology:
  assumes X≠0
  shows (PTopology X {X})={0,X}
proof
  {
    fix T
    assume T∈(PTopology X {X})
    with assms obtain M where M⊆{X}⋃M=T using Ptopology_is_a_topology(2)
      indiscrete_partition IsAbaseFor_def by auto
    then have T=0∨T=X by auto
  }
  then show (PTopology X {X})⊆{0,X} by auto
  from assms have 0∈(PTopology X {X}) using Ptopology_is_a_topology(1)
empty_open
    indiscrete_partition by auto
  moreover
  from assms have ⋃(PTopology X {X})∈(PTopology X {X}) using union_open
Ptopology_is_a_topology(1)
    indiscrete_partition by auto
  with assms have X∈(PTopology X {X}) using union_ptopology indiscrete_partition
    by auto
  ultimately show {0,X}⊆(PTopology X {X}) by auto
qed
```

The topological subspaces of the (PTopology X U) are partition topologies.

```
lemma subspace_ptopology:
  assumes U{is a partition of}X
  shows (PTopology X U) {restricted to} Y=(PTopology (X∩Y) ((U {restricted
to} Y)-{0}))
proof-
  from assms have U{is a base for}(PTopology X U) using Ptopology_is_a_topology(2)
    by auto
  then have (U{restricted to} Y){is a base for}(PTopology X U){restricted
to} Y
    using subspace_base_topology by auto
  then have ((U{restricted to} Y)-{0}){is a base for}(PTopology X U){restricted
```

```
to} Y using base_no_0
    by auto
  moreover
  from assms have ((U{restricted to} Y)-{0}) {is a partition of} (X∩Y)
    using restriction_partition by auto
  then have ((U{restricted to} Y)-{0}){is a base for}(PTopology (X∩Y)
((U {restricted to} Y)-{0}))
    using Ptopology_is_a_topology(2) by auto
  ultimately show thesis using same_base_same_top by auto
qed
```

79.8 Order topologies

79.9 Order topology is a topology

Given a totally ordered set, several topologies can be defined using the order relation. First we define an open interval, notice that the set defined as `Interval` is a closed interval; and open rays.

```
definition
  IntervalX where
  IntervalX(X,r,b,c)≡(Interval(r,b,c)∩X)-{b,c}
definition
  LeftRayX where
  LeftRayX(X,r,b)≡{c∈X. ⟨c,b⟩∈r}-{b}
definition
  RightRayX where
  RightRayX(X,r,b)≡{c∈X. ⟨b,c⟩∈r}-{b}
```

Intersections of intervals and rays.

```
lemma inter_two_intervals:
  assumes bu∈Xbv∈Xcu∈Xcv∈XIsLinOrder(X,r)
  shows IntervalX(X,r,bu,cu)∩IntervalX(X,r,bv,cv)=IntervalX(X,r,GreaterOf(r,bu,bv),Smaller(
proof
  have T:GreaterOf(r,bu,bv)∈XSmallerOf(r,cu,cv)∈X using assms
    GreaterOf_def SmallerOf_def by (cases ⟨bu,bv⟩∈r,simp,simp,cases ⟨cu,cv⟩∈r,simp,simp)
  {
    fix x
    assume ASS:x∈IntervalX(X,r,bu,cu)∩IntervalX(X,r,bv,cv)
    then have x∈IntervalX(X,r,bu,cu)x∈IntervalX(X,r,bv,cv) by auto
    then have BB:x∈Xx∈Interval(r,bu,cu)x≠bux≠cux∈Interval(r,bv,cv)x≠bvx≠cv
    using IntervalX_def assms by auto
    then have x∈X by auto
    moreover
    have x≠GreaterOf(r,bu,bv)x≠SmallerOf(r,cu,cv)
    proof-
      show x≠GreaterOf(r,bu,bv) using GreaterOf_def BB(6,3) by (cases
⟨bu,bv⟩∈r,simp+)
      show x≠SmallerOf(r,cu,cv) using SmallerOf_def BB(7,4) by (cases
⟨cu,cv⟩∈r,simp+)
```

```
    qed
    moreover
    have ⟨bu,x⟩∈r⟨x,cu⟩∈r⟨bv,x⟩∈r⟨x,cv⟩∈r using BB(2,5) Order_ZF_2_L1A
by auto
    then have ⟨GreaterOf(r,bu,bv),x⟩∈r⟨x,SmallerOf(r,cu,cv)⟩∈r using GreaterOf_def
SmallerOf_def
      by (cases ⟨bu,bv⟩∈r,simp,simp,cases ⟨cu,cv⟩∈r,simp,simp)
    then have x∈Interval(r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv)) us-
ing Order_ZF_2_L1 by auto
    ultimately
    have x∈IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv)) using
IntervalX_def T by auto
  }
  then show IntervalX(X, r, bu, cu) ∩ IntervalX(X, r, bv, cv) ⊆ IntervalX(X,
r, GreaterOf(r, bu, bv), SmallerOf(r, cu, cv))
    by auto
  {
    fix x
    assume x∈IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv))
    then have BB:x∈Xx∈Interval(r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv))x≠GreaterOf(r,bu,bv
    using IntervalX_def T by auto
    then have x∈X by auto
    moreover
    from BB(2) have CC:⟨GreaterOf(r,bu,bv),x⟩∈r⟨x,SmallerOf(r,cu,cv)⟩∈r
using Order_ZF_2_L1A by auto
    {
      {
        assume AS:⟨bu,bv⟩∈r
        then have GreaterOf(r,bu,bv)=bv using GreaterOf_def by auto
        then have ⟨bv,x⟩∈r using CC(1) by auto
        with AS have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms IsLinOrder_def trans_def
by (safe, blast)
      }
      moreover
      {
        assume AS:⟨bu,bv⟩∉r
        then have GreaterOf(r,bu,bv)=bu using GreaterOf_def by auto
        then have ⟨bu,x⟩∈r using CC(1) by auto
        from AS have ⟨bv,bu⟩∈r using assms IsLinOrder_def IsTotal_def
assms by auto
        with ‹⟨bu,x⟩∈r› have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms IsLinOrder_def
trans_def by (safe, blast)
      }
      ultimately have R:⟨bu,x⟩∈r ⟨bv,x⟩∈r by auto
      moreover
      {
        assume AS:x=bu
        then have ⟨bv,bu⟩∈r using R(2) by auto
        then have GreaterOf(r,bu,bv)=bu using GreaterOf_def assms IsLinOrder_def
```

```
          antisym_def by auto
          then have False using AS BB(3) by auto
        }
        moreover
        {
          assume AS:x=bv
          then have ⟨bu,bv⟩∈r using R(1) by auto
          then have GreaterOf(r,bu,bv)=bv using GreaterOf_def by auto
          then have False using AS BB(3) by auto
        }
        ultimately have ⟨bu,x⟩∈r ⟨bv,x⟩∈rx≠bux≠bv by auto
      }
      moreover
      {
        {
          assume AS:⟨cu,cv⟩∈r
          then have SmallerOf(r,cu,cv)=cu using SmallerOf_def by auto
          then have ⟨x,cu⟩∈r using CC(2) by auto
          with AS have ⟨x,cu⟩∈r ⟨x,cv⟩∈r using assms IsLinOrder_def trans_def
by(safe ,blast)
        }
        moreover
        {
          assume AS:⟨cu,cv⟩∉r
          then have SmallerOf(r,cu,cv)=cv using SmallerOf_def by auto
          then have ⟨x,cv⟩∈r using CC(2) by auto
          from AS have ⟨cv,cu⟩∈r using assms IsLinOrder_def IsTotal_def
by auto
          with ‹⟨x,cv⟩∈r› have ⟨x,cv⟩∈r ⟨x,cu⟩∈r using assms IsLinOrder_def
trans_def by(safe ,blast)
        }
        ultimately have R:⟨x,cv⟩∈r ⟨x,cu⟩∈r by auto
        moreover
        {
          assume AS:x=cv
          then have ⟨cv,cu⟩∈r using R(2) by auto
          then have SmallerOf(r,cu,cv)=cv using SmallerOf_def assms IsLinOrder_def
          antisym_def by auto
          then have False using AS BB(4) by auto
        }
        moreover
        {
          assume AS:x=cu
          then have ⟨cu,cv⟩∈r using R(1) by auto
          then have SmallerOf(r,cu,cv)=cu using SmallerOf_def by auto
          then have False using AS BB(4) by auto
        }
        ultimately have ⟨x,cu⟩∈r ⟨x,cv⟩∈rx≠cux≠cv by auto
      }
```

```
    ultimately
    have x∈IntervalX(X,r,bu,cu) x∈IntervalX(X,r,bv,cv) using Order_ZF_2_L1
IntervalX_def
      assms by auto
    then have x∈IntervalX(X, r, bu, cu) ∩ IntervalX(X, r, bv, cv)  by
auto
  }
  then show IntervalX(X,r,GreaterOf(r,bu,bv),SmallerOf(r,cu,cv)) ⊆ IntervalX(X,
r, bu, cu) ∩ IntervalX(X, r, bv, cv)
    by auto
qed

lemma inter_rray_interval:
  assumes bv∈Xbu∈Xcv∈XIsLinOrder(X,r)
  shows RightRayX(X,r,bu)∩IntervalX(X,r,bv,cv)=IntervalX(X,r,GreaterOf(r,bu,bv),cv)
proof
  {
    fix x
    assume x∈RightRayX(X,r,bu)∩IntervalX(X,r,bv,cv)
    then have x∈RightRayX(X,r,bu)x∈IntervalX(X,r,bv,cv) by auto
    then have BB:x∈Xx≠bux≠bvx≠cv⟨bu,x⟩∈rx∈Interval(r,bv,cv) using RightRayX_def
IntervalX_def
      by auto
    then have ⟨bv,x⟩∈r⟨x,cv⟩∈r using Order_ZF_2_L1A by auto
    with ‹⟨bu,x⟩∈r› have ⟨GreaterOf(r,bu,bv),x⟩∈r using GreaterOf_def
by (cases ⟨bu,bv⟩∈r,simp+)
    with ‹⟨x,cv⟩∈r› have x∈Interval(r,GreaterOf(r,bu,bv),cv) using Order_ZF_2_L1
by auto
    then have x∈IntervalX(X,r,GreaterOf(r,bu,bv),cv) using BB(1-4) IntervalX_def
GreaterOf_def
      by (simp)
  }
  then show RightRayX(X, r, bu) ∩ IntervalX(X, r, bv, cv) ⊆ IntervalX(X,
r, GreaterOf(r, bu, bv), cv) by auto
  {
    fix x
    assume x∈IntervalX(X, r, GreaterOf(r, bu, bv), cv)
    then have x∈Xx∈Interval(r,GreaterOf(r, bu, bv), cv)x≠cvx≠GreaterOf(r,
bu, bv) using IntervalX_def by auto
    then have R:⟨GreaterOf(r, bu, bv),x⟩∈r⟨x,cv⟩∈r using Order_ZF_2_L1A
by auto
    with ‹x≠cv› have ⟨x,cv⟩∈rx≠cv by auto
    moreover
    {
      {
        assume AS:⟨bu,bv⟩∈r
        then have GreaterOf(r,bu,bv)=bv using GreaterOf_def by auto
        then have ⟨bv,x⟩∈r using R(1) by auto
        with AS have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms unfolding IsLinOrder_def
```

```
trans_def by (safe,blast)
      }
      moreover
      {
        assume AS:⟨bu,bv⟩∉r
        then have GreaterOf(r,bu,bv)=bu using GreaterOf_def by auto
        then have ⟨bu,x⟩∈r using R(1) by auto
        from AS have ⟨bv,bu⟩∈r using assms unfolding IsLinOrder_def IsTotal_def
using assms by auto
        with ‹⟨bu,x⟩∈r› have ⟨bu,x⟩∈r ⟨bv,x⟩∈r using assms unfolding IsLinOrder_def
trans_def  by (safe,blast)
      }
      ultimately have T:⟨bu,x⟩∈r ⟨bv,x⟩∈r by auto
      moreover
      {
        assume AS:x=bu
        then have ⟨bv,bu⟩∈r using T(2) by auto
        then have GreaterOf(r,bu,bv)=bu unfolding GreaterOf_def using
assms unfolding IsLinOrder_def
        antisym_def by auto
        with ‹x≠GreaterOf(r,bu,bv)› have False using AS by auto
      }
      moreover
      {
        assume AS:x=bv
        then have ⟨bu,bv⟩∈r using T(1) by auto
        then have GreaterOf(r,bu,bv)=bv unfolding GreaterOf_def by auto
        with ‹x≠GreaterOf(r,bu,bv)› have False using AS by auto
      }
      ultimately have ⟨bu,x⟩∈r ⟨bv,x⟩∈rx≠bux≠bv by auto
    }
    with calculation ‹x∈X› have x∈RightRayX(X, r, bu)x∈IntervalX(X, r,
bv, cv) unfolding RightRayX_def IntervalX_def
      using Order_ZF_2_L1 by auto
    then have x∈RightRayX(X, r, bu) ∩ IntervalX(X, r, bv, cv)  by auto
  }
  then show IntervalX(X, r, GreaterOf(r, bu, bv), cv) ⊆ RightRayX(X,
r, bu) ∩ IntervalX(X, r, bv, cv)  by auto
qed


lemma inter_lray_interval:
  assumes bv∈Xcu∈Xcv∈XIsLinOrder(X,r)
  shows LeftRayX(X,r,cu)∩IntervalX(X,r,bv,cv)=IntervalX(X,r,bv,SmallerOf(r,cu,cv))
proof
  {
    fix x assume x∈LeftRayX(X,r,cu)∩IntervalX(X,r,bv,cv)
    then have B:x≠cux∈X⟨x,cu⟩∈r⟨bv,x⟩∈r⟨x,cv⟩∈rx≠bvx≠cv unfolding LeftRayX_def
IntervalX_def Interval_def
```

```
      by auto
    from ‹⟨x,cu⟩∈r› ‹⟨x,cv⟩∈r› have C:⟨x,SmallerOf(r, cu, cv)⟩∈r using
SmallerOf_def by (cases ⟨cu,cv⟩∈r,simp+)
    from B(7,1) have x≠SmallerOf(r,cu,cv) using SmallerOf_def by (cases
⟨cu,cv⟩∈r,simp+)
    then have x∈IntervalX(X,r,bv,SmallerOf(r,cu,cv)) using B C IntervalX_def
Order_ZF_2_L1 by auto
  }
  then show LeftRayX(X, r, cu) ∩ IntervalX(X, r, bv, cv) ⊆ IntervalX(X,
r, bv, SmallerOf(r, cu, cv)) by auto
  {
    fix x assume x∈IntervalX(X,r,bv,SmallerOf(r,cu,cv))
    then have R:x∈X⟨bv,x⟩∈r⟨x,SmallerOf(r,cu,cv)⟩∈rx≠bvx≠SmallerOf(r,cu,cv)
using IntervalX_def Interval_def
      by auto
    then have ⟨bv,x⟩∈rx≠bv by auto
    moreover
    {
      {
        assume AS:⟨cu,cv⟩∈r
        then have SmallerOf(r,cu,cv)=cu using SmallerOf_def by auto
        then have ⟨x,cu⟩∈r using R(3) by auto
        with AS have ⟨x,cu⟩∈r ⟨x,cv⟩∈r using assms unfolding IsLinOrder_def
trans_def by (safe, blast)
      }
      moreover
      {
        assume AS:⟨cu,cv⟩∉r
        then have SmallerOf(r,cu,cv)=cv using SmallerOf_def by auto
        then have ⟨x,cv⟩∈r using R(3) by auto
        from AS have ⟨cv,cu⟩∈r using assms IsLinOrder_def IsTotal_def
assms by auto
        with ‹⟨x,cv⟩∈r› have ⟨x,cv⟩∈r ⟨x,cu⟩∈r using assms IsLinOrder_def
trans_def by (safe, blast)
      }
      ultimately have T:⟨x,cv⟩∈r ⟨x,cu⟩∈r by auto
      moreover
      {
        assume AS:x=cu
        then have ⟨cu,cv⟩∈r using T(1) by auto
        then have SmallerOf(r,cu,cv)=cu using SmallerOf_def assms IsLinOrder_def
          antisym_def by auto
        with ‹x≠SmallerOf(r,cu,cv)› have False using AS by auto
      }
      moreover
      {
        assume AS:x=cv
        then have ⟨cv,cu⟩∈r using T(2) by auto
        then have SmallerOf(r,cu,cv)=cv using SmallerOf_def assms IsLinOrder_def
```

```
        antisym_def by auto
        with ‹x≠SmallerOf(r,cu,cv)› have False using AS by auto
      }
      ultimately have ⟨x,cu⟩∈r ⟨x,cv⟩∈rx≠cux≠cv by auto
    }
    with calculation ‹x∈X› have x∈LeftRayX(X,r,cu)x∈IntervalX(X, r, bv,
cv) using LeftRayX_def IntervalX_def Interval_def
      by auto
    then have x∈LeftRayX(X, r, cu) ∩ IntervalX(X, r, bv, cv) by auto
  }
  then show IntervalX(X, r, bv, SmallerOf(r, cu, cv)) ⊆ LeftRayX(X, r,
cu) ∩ IntervalX(X, r, bv, cv)  by auto
qed

lemma inter_lray_rray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
  shows LeftRayX(X,r,bu)∩RightRayX(X,r,cv)=IntervalX(X,r,cv,bu)
  unfolding LeftRayX_def RightRayX_def IntervalX_def Interval_def by auto

lemma inter_lray_lray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
  shows LeftRayX(X,r,bu)∩LeftRayX(X,r,cv)=LeftRayX(X,r,SmallerOf(r,bu,cv))
proof
  {
    fix x
    assume x∈LeftRayX(X,r,bu)∩LeftRayX(X,r,cv)
    then have B:x∈X⟨x,bu⟩∈r⟨x,cv⟩∈rx≠bux≠cv using LeftRayX_def by auto
    then have C:⟨x,SmallerOf(r,bu,cv)⟩∈r using SmallerOf_def by (cases
⟨bu,cv⟩∈r, auto)
    from B have D:x≠SmallerOf(r,bu,cv) using SmallerOf_def by (cases
⟨bu,cv⟩∈r, auto)
    from B C D have x∈LeftRayX(X,r,SmallerOf(r,bu,cv)) using LeftRayX_def
by auto
  }
  then show LeftRayX(X, r, bu) ∩ LeftRayX(X, r, cv) ⊆ LeftRayX(X, r,
SmallerOf(r, bu, cv)) by auto
  {
    fix x
    assume x∈LeftRayX(X, r, SmallerOf(r, bu, cv))
    then have R:x∈X⟨x,SmallerOf(r,bu,cv)⟩∈rx≠SmallerOf(r,bu,cv) using
LeftRayX_def by auto
    {
      {
        assume AS:⟨bu,cv⟩∈r
        then have SmallerOf(r,bu,cv)=bu using SmallerOf_def by auto
        then have ⟨x,bu⟩∈r using R(2) by auto
        with AS have ⟨x,bu⟩∈r ⟨x,cv⟩∈r using assms IsLinOrder_def trans_def
by(safe, blast)
      }
```

```
      moreover
      {
        assume AS:⟨bu,cv⟩∉r
        then have SmallerOf(r,bu,cv)=cv using SmallerOf_def by auto
        then have ⟨x,cv⟩∈r using R(2) by auto
        from AS have ⟨cv,bu⟩∈r using assms IsLinOrder_def IsTotal_def
assms by auto
        with ‹⟨x,cv⟩∈r› have ⟨x,cv⟩∈r ⟨x,bu⟩∈r using assms IsLinOrder_def
trans_def by(safe, blast)
      }
      ultimately have T:⟨x,cv⟩∈r ⟨x,bu⟩∈r by auto
      moreover
      {
        assume AS:x=bu
        then have ⟨bu,cv⟩∈r using T(1) by auto
        then have SmallerOf(r,bu,cv)=bu using SmallerOf_def assms IsLinOrder_def
          antisym_def by auto
        with ‹x≠SmallerOf(r,bu,cv)› have False using AS by auto
      }
      moreover
      {
        assume AS:x=cv
        then have ⟨cv,bu⟩∈r using T(2) by auto
        then have SmallerOf(r,bu,cv)=cv using SmallerOf_def assms IsLinOrder_def
          antisym_def by auto
        with ‹x≠SmallerOf(r,bu,cv)› have False using AS by auto
      }
      ultimately have ⟨x,bu⟩∈r ⟨x,cv⟩∈rx≠bux≠cv by auto
    }
    with ‹x∈X› have x∈ LeftRayX(X, r, bu) ∩ LeftRayX(X, r, cv) using
LeftRayX_def by auto
  }
  then show LeftRayX(X, r, SmallerOf(r, bu, cv)) ⊆ LeftRayX(X, r, bu)
∩ LeftRayX(X, r, cv)  by auto
qed

lemma inter_rray_rray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
  shows RightRayX(X,r,bu)∩RightRayX(X,r,cv)=RightRayX(X,r,GreaterOf(r,bu,cv))
proof
  {
    fix x
    assume x∈RightRayX(X,r,bu)∩RightRayX(X,r,cv)
    then have B:x∈X⟨bu,x⟩∈r⟨cv,x⟩∈rx≠bux≠cv using RightRayX_def by auto
    then have C:⟨GreaterOf(r,bu,cv),x⟩∈r using GreaterOf_def by (cases
⟨bu,cv⟩∈r,auto)
    from B have D:x≠GreaterOf(r,bu,cv) using GreaterOf_def by (cases
⟨bu,cv⟩∈r,auto)
    from B C D have x∈RightRayX(X,r,GreaterOf(r,bu,cv)) using RightRayX_def
```

```
by auto
  }
  then show  RightRayX(X, r, bu) ∩ RightRayX(X, r, cv) ⊆ RightRayX(X,
r, GreaterOf(r, bu, cv)) by auto
  {
    fix x
    assume x∈RightRayX(X, r, GreaterOf(r, bu, cv))
    then have R:x∈X⟨GreaterOf(r,bu,cv),x⟩∈rx≠GreaterOf(r,bu,cv) using
RightRayX_def by auto
    {
      {
        assume AS:⟨bu,cv⟩∈r
        then have GreaterOf(r,bu,cv)=cv using GreaterOf_def by auto
        then have ⟨cv,x⟩∈r using R(2) by auto
        with AS have ⟨bu,x⟩∈r ⟨cv,x⟩∈r using assms IsLinOrder_def trans_def
by(safe, blast)
      }
      moreover
      {
        assume AS:⟨bu,cv⟩∉r
        then have GreaterOf(r,bu,cv)=bu using GreaterOf_def by auto
        then have ⟨bu,x⟩∈r using R(2) by auto
        from AS have ⟨cv,bu⟩∈r using assms IsLinOrder_def IsTotal_def
assms by auto
        with ‹⟨bu,x⟩∈r› have ⟨cv,x⟩∈r ⟨bu,x⟩∈r using assms IsLinOrder_def
trans_def by(safe, blast)
      }
      ultimately have T:⟨cv,x⟩∈r ⟨bu,x⟩∈r by auto
      moreover
      {
        assume AS:x=bu
        then have ⟨cv,bu⟩∈r using T(1) by auto
        then have GreaterOf(r,bu,cv)=bu using GreaterOf_def assms IsLinOrder_def
          antisym_def by auto
        with ‹x≠GreaterOf(r,bu,cv)› have False using AS by auto
      }
      moreover
      {
        assume AS:x=cv
        then have ⟨bu,cv⟩∈r using T(2) by auto
        then have GreaterOf(r,bu,cv)=cv using GreaterOf_def assms IsLinOrder_def
          antisym_def by auto
        with ‹x≠GreaterOf(r,bu,cv)› have False using AS by auto
      }
      ultimately have ⟨bu,x⟩∈r ⟨cv,x⟩∈rx≠bux≠cv by auto
    }
    with ‹x∈X› have x∈ RightRayX(X, r, bu) ∩ RightRayX(X, r, cv)  us-
ing RightRayX_def by auto
  }
```

```
  then show RightRayX(X, r, GreaterOf(r, bu, cv)) ⊆ RightRayX(X, r, bu)
∩ RightRayX(X, r, cv) by auto
qed
```

The open intervals and rays satisfy the base condition.

```
lemma intervals_rays_base_condition:
  assumes IsLinOrder(X,r)
  shows {IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X} {satisfies the base condition}
proof-
  let I={IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}
  let R={RightRayX(X,r,b). b∈X}
  let L={LeftRayX(X,r,b). b∈X}
  let B={IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X}
  {
    fix U V
    assume A:U∈BV∈B
    then have dU:U∈I∨U∈L∨U∈Rand dV:V∈I∨V∈L∨V∈R by auto
    {
      assume S:V∈I
      {
        assume U∈I
        with S obtain bu cu bv cv where A:U=IntervalX(X,r,bu,cu)V=IntervalX(X,r,bv,cv)bu∈X
          by auto
        then have SmallerOf(r,cu,cv)∈XGreaterOf(r,bu,bv)∈X by (cases
⟨cu,cv⟩∈r,simp add:SmallerOf_def A,simp add:SmallerOf_def A,
          cases ⟨bu,bv⟩∈r,simp add:GreaterOf_def A,simp add:GreaterOf_def
A)
        with A have U∩V∈B using inter_two_intervals assms by auto
      }
      moreover
      {
        assume U∈L
        with S obtain bu bv cv where A:U=LeftRayX(X, r,bu)V=IntervalX(X,r,bv,cv)bu∈Xbv∈Xcv
        by auto
        then have SmallerOf(r,bu,cv)∈X using SmallerOf_def by (cases
⟨bu,cv⟩∈r,auto)
        with A have U∩V∈B using inter_lray_interval assms by auto
      }
      moreover
      {
        assume U∈R
        with S obtain cu bv cv where A:U=RightRayX(X,r,cu)V=IntervalX(X,r,bv,cv)cu∈Xbv∈Xcv
        by auto
        then have GreaterOf(r,cu,bv)∈X using GreaterOf_def by (cases
⟨cu,bv⟩∈r,auto)
        with A have U∩V∈B using inter_rray_interval assms by auto
      }
```

```
      ultimately have U∩V∈B using dU by auto
    }
    moreover
    {
      assume S:V∈L
      {
        assume U∈I
        with S obtain bu bv cv where A:V=LeftRayX(X, r,bu)U=IntervalX(X,r,bv,cv)bu∈Xbv∈Xcv
          by auto
        then have SmallerOf(r,bu,cv)∈X using SmallerOf_def by (cases
⟨bu,cv⟩∈r, auto)
        have U∩V=V∩U by auto
        with A ‹SmallerOf(r,bu,cv)∈X› have U∩V∈B using inter_lray_interval
assms by auto
      }
      moreover
      {
        assume U∈R
        with S obtain bu cv where A:V=LeftRayX(X,r,bu)U=RightRayX(X,r,cv)bu∈Xcv∈X
        by auto
        have U∩V=V∩U by auto
        with A have U∩V∈B using inter_lray_rray assms by auto
      }
      moreover
      {
        assume U∈L
        with S obtain bu bv where A:U=LeftRayX(X,r,bu)V=LeftRayX(X,r,bv)bu∈Xbv∈X
        by auto
        then have SmallerOf(r,bu,bv)∈X using SmallerOf_def by (cases
⟨bu,bv⟩∈r, auto)
        with A have U∩V∈B using inter_lray_lray assms by auto
      }
      ultimately have U∩V∈B using dU by auto
    }
    moreover
    {
      assume S:V∈R
      {
        assume U∈I
        with S obtain cu bv cv where A:V=RightRayX(X,r,cu)U=IntervalX(X,r,bv,cv)cu∈Xbv∈Xcv
        by auto
        then have GreaterOf(r,cu,bv)∈X using GreaterOf_def by (cases
⟨cu,bv⟩∈r,auto)
        have U∩V=V∩U by auto
        with A ‹GreaterOf(r,cu,bv)∈X› have U∩V∈B using inter_rray_interval
assms by auto
      }
      moreover
      {
```

```
        assume U∈L
        with S obtain bu cv where A:U=LeftRayX(X,r,bu)V=RightRayX(X,r,cv)bu∈Xcv∈X
        by auto
        then have U∩V∈B using inter_lray_rray assms by auto
      }
      moreover
      {
        assume U∈R
        with S obtain cu cv where A:U=RightRayX(X,r,cu)V=RightRayX(X,r,cv)cu∈Xcv∈X
        by auto
        then have GreaterOf(r,cu,cv)∈X using GreaterOf_def by (cases
⟨cu,cv⟩∈r,auto)
        with A have U∩V∈B using inter_rray_rray assms by auto
      }
      ultimately have U∩V∈B using dU by auto
    }
    ultimately have  S:U∩V∈B using dV by auto
    {
      fix x
      assume x∈U∩V
      then have x∈U∩V∧U∩V⊆U∩V by auto
      then have ∃W. W∈(B)∧ x∈W ∧ W ⊆ U∩V using S by blast
      then have ∃W∈(B). x∈W ∧ W ⊆ U∩V by blast
    }
    hence (∀x ∈ U∩V. ∃W∈(B). x∈W ∧ W ⊆ U∩V) by auto
  }
  then show thesis using SatisfiesBaseCondition_def by auto
qed
```

Since the intervals and rays form a base of a topology, and this topology is uniquely determined; we can built it. In the definition we have to make sure that we have a totally ordered set.

```
definition
  OrderTopology (OrdTopology _ _ 50) where
  IsLinOrder(X,r) ⟹ OrdTopology X r ≡ TopologyBase {IntervalX(X,r,b,c).
⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b). b∈X}

theorem Ordtopology_is_a_topology:
  assumes IsLinOrder(X,r)
  shows (OrdTopology X r) {is a topology} and {IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,
b∈X}∪{RightRayX(X,r,b). b∈X} {is a base for} (OrdTopology X r)
  using assms Base_topology_is_a_topology intervals_rays_base_condition

    OrderTopology_def by auto

lemma topology0_ordtopology:
  assumes IsLinOrder(X,r)
  shows topology0(OrdTopology X r)
  using Ordtopology_is_a_topology topology0_def assms by auto
```

79.10 Total set

The topology is defined in the set X, when X has more than one point

```
lemma union_ordtopology:
  assumes IsLinOrder(X,r)∃x y. x≠y ∧ x∈X∧ y∈X
  shows ⋃(OrdTopology X r)=X
proof
  let B={IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X}
  have base:B {is a base for} (OrdTopology X r) using Ordtopology_is_a_topology(2)
assms(1)
    by auto
  from assms(2) obtain x y where T:x≠y ∧ x∈X∧ y∈X by auto
  then have B:x∈LeftRayX(X,r,y)∨x∈RightRayX(X,r,y) using LeftRayX_def
RightRayX_def
    assms(1) IsLinOrder_def IsTotal_def by auto
  then have x∈⋃B using T by auto
  then have x:x∈⋃(OrdTopology X r) using Top_1_2_L5 base by auto
  {
    fix z
    assume z:z∈X
    {
      assume x=z
      then have z∈⋃(OrdTopology X r) using x by auto
    }
    moreover
    {
      assume x≠z
      with z T have z∈LeftRayX(X,r,x)∨z∈RightRayX(X,r,x)x∈X using LeftRayX_def
RightRayX_def
        assms(1) IsLinOrder_def IsTotal_def by auto
      then have z∈⋃B by auto
      then have z∈⋃(OrdTopology X r) using Top_1_2_L5 base by auto
    }
    ultimately have z∈⋃(OrdTopology X r) by auto
  }
  then show X⊆⋃(OrdTopology X r) by auto
  have ⋃B⊆X using IntervalX_def LeftRayX_def RightRayX_def by auto
  then show ⋃(OrdTopology X r)⊆X using Top_1_2_L5 base by auto
qed
```

The interior, closure and boundary can be calculated using the formulas proved in the section that deals with the base.

The subspace of an order topology doesn't have to be an order topology.

79.11 Right order and Left order topologies.

Notice that the left and right rays are closed under intersection, hence they form a base of a topology. They are called right order topology and left order topology respectively.

If the order in X has a minimal or a maximal element, is necessary to consider X as an element of the base or that limit point wouldn't be in any basic open set.

79.11.1 Right and Left Order topologies are topologies

```
lemma leftrays_base_condition:
assumes IsLinOrder(X,r)
shows {LeftRayX(X,r,b). b∈X}∪{X} {satisfies the base condition}
proof-
  {
    fix U V
    assume U∈{LeftRayX(X,r,b). b∈X}∪{X}V∈{LeftRayX(X,r,b). b∈X}∪{X}
    then obtain b c where A:(b∈X∧U=LeftRayX(X,r,b))∨U=X(c∈X∧V=LeftRayX(X,r,c))∨V=XU⊆XV⊆X
    unfolding LeftRayX_def by auto
    then have (U∩V=LeftRayX(X,r,SmallerOf(r,b,c))∧b∈X∧c∈X)∨U∩V=X∨(U∩V=LeftRayX(X,r,c)∧c∈
      using inter_lray_lray assms by auto
    moreover
    have b∈X∧c∈X ⟶ SmallerOf(r,b,c)∈X unfolding SmallerOf_def by (cases
⟨b,c⟩∈r,auto)
    ultimately have U∩V∈{LeftRayX(X,r,b). b∈X}∪{X} by auto
    hence ∀x∈U∩V. ∃W∈{LeftRayX(X,r,b). b∈X}∪{X}. x∈W∧W⊆U∩V by blast
  }
  moreover
  then show thesis using SatisfiesBaseCondition_def by auto
qed

lemma rightrays_base_condition:
assumes IsLinOrder(X,r)
shows {RightRayX(X,r,b). b∈X}∪{X} {satisfies the base condition}
proof-
  {
    fix U V
    assume U∈{RightRayX(X,r,b). b∈X}∪{X}V∈{RightRayX(X,r,b). b∈X}∪{X}
    then obtain b c where A:(b∈X∧U=RightRayX(X,r,b))∨U=X(c∈X∧V=RightRayX(X,r,c))∨V=XU⊆XV
    unfolding RightRayX_def by auto
    then have (U∩V=RightRayX(X,r,GreaterOf(r,b,c))∧b∈X∧c∈X)∨U∩V=X∨(U∩V=RightRayX(X,r,c)∧

      using inter_rray_rray assms by auto
    moreover
    have b∈X∧c∈X ⟶ GreaterOf(r,b,c)∈X using GreaterOf_def by (cases
⟨b,c⟩∈r,auto)
    ultimately have U∩V∈{RightRayX(X,r,b). b∈X}∪{X} by auto
```

```
    hence ∀x∈U∩V. ∃W∈{RightRayX(X,r,b). b∈X}∪{X}. x∈W∧W⊆U∩V by blast
  }
  then show thesis using SatisfiesBaseCondition_def by auto
qed
```

```
definition
  LeftOrderTopology (LOrdTopology _ _ 50) where
  IsLinOrder(X,r) ⟹ LOrdTopology X r ≡ TopologyBase {LeftRayX(X,r,b).
b∈X}∪{X}
```

```
definition
  RightOrderTopology (ROrdTopology _ _ 50) where
  IsLinOrder(X,r) ⟹ ROrdTopology X r ≡ TopologyBase {RightRayX(X,r,b).
b∈X}∪{X}
```

```
theorem LOrdtopology_ROrdtopology_are_topologies:
  assumes IsLinOrder(X,r)
  shows (LOrdTopology X r) {is a topology} and {LeftRayX(X,r,b). b∈X}∪{X}
{is a base for} (LOrdTopology X r)
  and (ROrdTopology X r) {is a topology} and {RightRayX(X,r,b). b∈X}∪{X}
{is a base for} (ROrdTopology X r)
  using Base_topology_is_a_topology leftrays_base_condition assms rightrays_base_condition
     LeftOrderTopology_def RightOrderTopology_def by auto
```

```
lemma topology0_lordtopology_rordtopology:
  assumes IsLinOrder(X,r)
  shows  topology0(LOrdTopology X r) and topology0(ROrdTopology X r)
  using LOrdtopology_ROrdtopology_are_topologies topology0_def assms by
auto
```

79.11.2 Total set

The topology is defined on the set X

```
lemma union_lordtopology_rordtopology:
  assumes IsLinOrder(X,r)
  shows ⋃(LOrdTopology X r)=X and ⋃(ROrdTopology X r)=X
  using Top_1_2_L5[OF LOrdtopology_ROrdtopology_are_topologies(2)[OF assms]]
    Top_1_2_L5[OF LOrdtopology_ROrdtopology_are_topologies(4)[OF assms]]
  unfolding LeftRayX_def RightRayX_def by auto
```

79.12 Union of Topologies

The union of two topologies is not a topology. A way to overcome this fact is to define the following topology:

```
definition
  joinT (joinT _ 90) where
  (∀T∈M. T{is a topology} ∧ (∀Q∈M. ⋃Q=⋃T)) ⟹ (joinT M ≡ THE T. (⋃M){is
a subbase for} T)
```

First let's proof that given a family of sets, then it is a subbase for a topology.

The first result states that from any family of sets we get a base using finite intersections of them. The second one states that any family of sets is a subbase of some topology.

```
theorem subset_as_subbase:
  shows {⋂A. A ∈ FinPow(B)} {satisfies the base condition}
proof-
  {
    fix U V
    assume A:U∈{⋂A. A ∈ FinPow(B)} ∧ V∈{⋂A. A ∈ FinPow(B)}
    then obtain M R where MR:Finite(M)Finite(R)M⊆BR⊆B
    U=⋂MV=⋂R
    using FinPow_def by auto
    {
      fix x
      assume AS:x∈U∩V
      then have N:M≠0R≠0 using MR(5,6) by auto
      have Finite(M ∪R) using MR(1,2) by auto
      moreover
      have M ∪ R∈Pow(B) using MR(3,4) by auto
      ultimately have M∪R∈FinPow(B) using FinPow_def by auto
      then have ⋂(M∪R)∈{⋂A. A ∈ FinPow(B)} by auto
      moreover
      from N have ⋂(M∪R)⊆⋂M⋂(M∪R)⊆⋂R by auto
      then have ⋂(M∪R)⊆U∩V using MR(5,6) by auto
      moreover
      {
        fix S
        assume S∈M ∪ R
        then have S∈M∨S∈R by auto
        then have x∈S using AS MR(5,6) by auto
      }
      then have x∈⋂(M ∪ R) using N by auto
      ultimately have ∃W∈{⋂A. A ∈ FinPow(B)}. x∈W∧W⊆U∩V by blast
    }
    then have (∀x ∈ U∩V. ∃W∈{⋂A. A ∈ FinPow(B)}. x∈W ∧ W ⊆ U∩V) by
auto
  }
  then have ∀U V. ((U∈{⋂A. A ∈ FinPow(B)} ∧ V∈{⋂A. A ∈ FinPow(B)})
⟶
    (∀x ∈ U∩V. ∃W∈{⋂A. A ∈ FinPow(B)}. x∈W ∧ W ⊆ U∩V)) by auto
  then show {⋂A. A ∈ FinPow(B)} {satisfies the base condition}
    using SatisfiesBaseCondition_def by auto
qed

theorem Top_subbase:
  assumes T = {⋃A. A∈Pow({⋂A. A ∈ FinPow(B)})}
  shows T {is a topology} and B {is a subbase for} T
```

```
proof-
  {
    fix S
    assume S∈B
    then have {S}∈FinPow(B)⋂{S}=S using FinPow_def by auto
    then have {S}∈Pow({⋂A. A ∈ FinPow(B)}) by (blast+)
    then have ⋃{S}∈{⋃A. A∈Pow({⋂A. A ∈ FinPow(B)})} by blast
    then have S∈{⋃A. A∈Pow({⋂A. A ∈ FinPow(B)})} by auto
    then have S∈T using assms by auto
  }
  then have B⊆T by auto
  moreover
  have {⋂A. A ∈ FinPow(B)} {satisfies the base condition}
    using subset_as_subbase by auto
  then have T {is a topology} and {⋂A. A ∈ FinPow(B)} {is a base for}
T
    using Top_1_2_T1 assms by auto
  ultimately show T {is a topology} and B{is a subbase for}T
    using IsAsubBaseFor_def by auto
qed
```

A subbase defines a unique topology.

```
theorem same_subbase_same_top:
  assumes B {is a subbase for} T and B {is a subbase for} S
  shows T = S
  using IsAsubBaseFor_def assms same_base_same_top
  by auto

end
```

80 Properties in Topology

```
theory Topology_ZF_properties imports Topology_ZF_examples Topology_ZF_examples_1

begin
```

This theory deals with topological properties which make use of cardinals.

80.1 Properties of compactness

It is already defined what is a compact topological space, but the is a generalization which may be useful sometimes.

```
definition
  IsCompactOfCard (_{is compact of cardinal}_ {in}_ 90)
  where K{is compact of cardinal} Q{in}T ≡ (Card(Q) ∧ K ⊆ ⋃T ∧
  (∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺Q)))
```

The usual compact property is the one defined over the cardinal of the natural numbers.

```
lemma Compact_is_card_nat:
  shows K{is compact in}T ⟷ (K{is compact of cardinal} nat {in}T)
proof
  {
    assume K{is compact in}T
    then have sub:K ⊆ ⋃T  and reg:(∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈
FinPow(M). K ⊆ ⋃N))
      using IsCompact_def by auto
    {
      fix M
      assume M∈Pow(T)K⊆⋃M
      with reg obtain N where N∈FinPow(M) K⊆⋃N by blast
      then have Finite(N) using FinPow_def by auto
      then obtain n where A:n∈natN ≈n using Finite_def by auto
      from A(1) have n≺nat using n_lesspoll_nat by auto
      with A(2) have N≲nat using lesspoll_def eq_lepoll_trans by auto
      moreover
      {
        assume N ≈nat
        then have nat ≈ N using eqpoll_sym by auto
        with A(2) have nat ≈n using eqpoll_trans by blast
        then have n ≈nat using eqpoll_sym by auto
        with ‹n≺nat› have False using lesspoll_def by auto
      }
      then have ~(N ≈nat) by auto
      with calculation ‹K⊆⋃N›‹N∈FinPow(M)› have N≺natK⊆⋃NN∈Pow(M)
using lesspoll_def
        FinPow_def by auto
      hence (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺nat) by auto
    }
    with sub show K{is compact of cardinal} nat {in}T using IsCompactOfCard_def
Card_nat by auto
  }
  {
    assume (K{is compact of cardinal} nat {in}T)
    then have sub:K⊆⋃T and reg:(∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N ∈ Pow(M).
K ⊆ ⋃N ∧ N≺nat))
      using IsCompactOfCard_def by auto
    {
      fix M
      assume M∈Pow(T)K⊆⋃M
      with reg have (∃ N ∈ Pow(M). K ⊆ ⋃N ∧ N≺nat) by auto
      then obtain N where N∈Pow(M)K⊆⋃NN≺nat by blast
      then have N∈FinPow(M)K⊆⋃N using lesspoll_nat_is_Finite FinPow_def
by auto
      hence ∃N∈FinPow(M). K⊆⋃N by auto
    }
```

```
    with sub show K{is compact in}T using IsCompact_def by auto
  }
qed
```

Another property of this kind widely used is the Lindeloef property; it is the one on the successor of the natural numbers.

```
definition
  IsLindeloef  (_{is lindeloef in}_ 90) where
  K {is lindeloef in} T≡K{is compact of cardinal}csucc(nat){in}T
```

It would be natural to think that every countable set with any topology is Lindeloef; but this statement is not provable in ZF. The reason is that to build a subcover, most of the time we need to *choose* sets from an infinite collection which cannot be done in ZF. Additional axioms are needed, but strictly weaker than the axiom of choice.

However, if the topology has not many open sets, then the topological space is indeed compact.

```
theorem card_top_comp:
  assumes Card(Q) T≺Q K⊆⋃T
  shows (K){is compact of cardinal}Q{in}T
proof-
  {
    fix M assume M:M⊆T K⊆⋃M
    from M(1) assms(2) have M≺Q using subset_imp_lepoll lesspoll_trans1
by blast
    with M(2) have ∃N∈Pow(M). K⊆⋃N ∧ N≺Q by auto
  }
  with assms(1,3) show thesis unfolding IsCompactOfCard_def by auto
qed
```

The union of two compact sets, is compact; of any cardinality.

```
theorem union_compact:
  assumes K{is compact of cardinal}Q{in}T K1{is compact of cardinal}Q{in}T
InfCard(Q)
  shows (K ∪ K1){is compact of cardinal}Q{in}T unfolding IsCompactOfCard_def
proof(safe)
  from assms(1) show Card(Q) unfolding IsCompactOfCard_def by auto
  fix x assume x∈K then show x∈⋃T using assms(1) unfolding IsCompactOfCard_def
by blast
next
  fix x assume x∈K1 then show x∈⋃T using assms(2) unfolding IsCompactOfCard_def
by blast
next
  fix M assume M⊆T K∪K1⊆⋃M
  then have K⊆⋃MK1⊆⋃M by auto
  with ‹M⊆T› have ∃N∈Pow(M). K ⊆ ⋃N ∧ N ≺ Q∃N∈Pow(M). K1 ⊆ ⋃N ∧
N ≺ Q using assms unfolding IsCompactOfCard_def
```

```
    by auto
  then obtain NK NK1 where NK∈Pow(M)NK1∈Pow(M)K ⊆ ⋃NKK1 ⊆ ⋃NK1NK ≺
QNK1 ≺ Q by auto
  then have NK∪NK1 ≺ QK∪K1⊆⋃(NK∪NK1)NK∪NK1∈Pow(M) using assms(3) less_less_imp_un_less
by auto
  then show ∃N∈Pow(M). K∪K1⊆⋃N ∧ N≺Q by auto
qed
```

If a set is compact of cardinality Q for some topology, it is compact of cardinality Q for every coarser topology.

```
theorem compact_coarser:
  assumes T1⊆T and ⋃T1=⋃T and (K){is compact of cardinal}Q{in}T
  shows (K){is compact of cardinal}Q{in}T1
proof-
  {
    fix M
    assume AS:M∈Pow(T1)K⊆⋃M
    then have M∈Pow(T)K⊆⋃M using assms(1) by auto
    then have ∃N∈Pow(M).K⊆⋃N∧N≺Q using assms(3) unfolding IsCompactOfCard_def
by auto
  }
  then show (K){is compact of cardinal}Q{in}T1 using assms(3,2) unfold-
ing IsCompactOfCard_def by auto
qed
```

If some set is compact for some cardinal, it is compact for any greater cardinal.

```
theorem compact_greater_card:
  assumes Q≲Q1 and (K){is compact of cardinal}Q{in}T and Card(Q1)
  shows (K){is compact of cardinal}Q1{in}T
proof-
  {
    fix M
    assume AS: M∈Pow(T)K⊆⋃M
    then have ∃N∈Pow(M).K⊆⋃N∧N≺Q using assms(2) unfolding IsCompactOfCard_def
by auto
    then have ∃N∈Pow(M).K⊆⋃N∧N≺Q1 using assms(1) lesspoll_trans2
      unfolding IsCompactOfCard_def by auto
  }
  then show thesis using assms(2,3) unfolding IsCompactOfCard_def by
auto
qed
```

A closed subspace of a compact space of any cardinality, is also compact of the same cardinality.

```
theorem compact_closed:
  assumes K {is compact of cardinal} Q {in} T
    and R {is closed in} T
```

```
  shows (K∩R) {is compact of cardinal} Q {in} T
proof-
  {
    fix M
    assume AS:M∈Pow(T)K∩R⊆⋃M
    have ⋃T-R∈T using assms(2) IsClosed_def by auto
    have K-R⊆(⋃T-R) using assms(1) IsCompactOfCard_def by auto
    with ‹⋃T-R∈T› have K⊆⋃(M ∪{⋃T-R}) and M ∪{⋃T-R}∈Pow(T)
    proof (safe)
      {
        fix x
        assume x∈M
        with AS(1) show x∈T by auto
      }
      {
        fix x
        assume x∈K
        have x∈R∨x∉R by auto
        with ‹x∈K› have x∈K∩R∨x∈K-R by auto
        with AS(2) ‹K-R⊆(⋃T-R)› have x∈⋃M∨x∈(⋃T-R) by auto
        then show x∈⋃(M ∪{⋃T-R}) by auto
      }
    qed
    with assms(1) have ∃N∈Pow(M∪{⋃T-R}). K ⊆ ⋃N ∧ N ≺ Q unfolding
IsCompactOfCard_def by auto
    then obtain N where cub:N∈Pow(M∪{⋃T-R}) K⊆⋃N N≺Q by auto
    have N-{⋃T-R}∈Pow(M)K∩R⊆⋃(N-{⋃T-R}) N-{⋃T-R}≺Q
    proof (safe)
      {
        fix x
        assume x∈Nx∉M
        then show x=⋃T-R using cub(1) by auto
      }
      {
        fix x
        assume x∈Kx∈R
        then have x∉⋃T-Rx∈K by auto
        then show x∈⋃(N-{⋃T-R}) using cub(2) by blast
      }
      have N-{⋃T-R}⊆N by auto
      with cub(3) show N-{⋃T-R}≺Q using subset_imp_lepoll lesspoll_trans1
by blast
    qed
    then have ∃N∈Pow(M). K∩R⊆⋃N ∧ N≺Q by auto
  }
  then have ∀M∈Pow(T). (K ∩ R ⊆ ⋃M ⟶ (∃N∈Pow(M). K ∩ R ⊆ ⋃N ∧ N
≺ Q)) by auto
  then show thesis using IsCompactOfCard_def assms(1) by auto
qed
```

80.2 Properties of numerability

The properties of numerability deal with cardinals of some sets built from the topology. The properties which are normally used are the ones related to the cardinal of the natural numbers or its successor.

```
definition
  IsFirstOfCard (_ {is of first type of cardinal}_ 90) where
  (T {is of first type of cardinal} Q) ≡ ∀x∈⋃T. (∃B. (B {is a base for}
T) ∧ ({b∈B. x∈b} ≺ Q))
```

```
definition
  IsSecondOfCard (_ {is of second type of cardinal}_ 90) where
  (T {is of second type of cardinal}Q) ≡ (∃B. (B {is a base for} T) ∧
(B ≺ Q))
```

```
definition
  IsSeparableOfCard (_{is separable of cardinal}_ 90) where
  T{is separable of cardinal}Q≡ ∃U∈Pow(⋃T). Closure(U,T)=⋃T ∧ U≺Q
```

```
definition
  IsFirstCountable (_ {is first countable} 90) where
  (T {is first countable}) ≡ T {is of first type of cardinal} csucc(nat)
```

```
definition
  IsSecondCountable (_ {is second countable} 90) where
  (T {is second countable}) ≡ (T {is of second type of cardinal}csucc(nat))
```

```
definition
  IsSeparable (_{is separable} 90) where
  T{is separable}≡ T{is separable of cardinal}csucc(nat)
```

If a set is of second type of cardinal `Q`, then it is of first type of that same cardinal.

```
theorem second_imp_first:
  assumes T{is of second type of cardinal}Q
  shows T{is of first type of cardinal}Q
proof-
  from assms have ∃B. (B {is a base for} T) ∧ (B ≺ Q) using IsSecondOfCard_def
by auto
  then obtain B where base:(B {is a base for} T) ∧ (B ≺ Q) by auto
  {
    fix x
    assume x∈⋃T
    have {b∈B. x∈b}⊆B by auto
    then have {b∈B. x∈b}≲B using subset_imp_lepoll by auto
    with base have {b∈B. x∈b}≺Q using lesspoll_trans1 by auto
    with base have (B {is a base for} T) ∧ {b∈B. x∈b}≺Q by auto
  }
  then have ∀x∈⋃T. ∃B. (B {is a base for} T) ∧ {b∈B. x∈b}≺Q by auto
```

```
  then show thesis using IsFirstOfCard_def by auto
qed
```

A set is dense iff it intersects all non-empty, open sets of the topology.

```
lemma dense_int_open:
  assumes T{is a topology} and A⊆⋃T
  shows Closure(A,T)=⋃T ⟷ (∀U∈T. U≠0 ⟶ A∩U≠0)
proof
  assume AS:Closure(A,T)=⋃T
  {
    fix U
    assume Uopen:U∈T and U≠0
    then have U∩⋃T≠0 by auto
    with AS have U∩Closure(A,T) ≠0 by auto
    with assms Uopen have U∩A≠0 using topology0.cl_inter_neigh topology0_def
by blast
  }
  then show ∀U∈T. U≠0 ⟶ A∩U≠0 by auto
  next
  assume AS:∀U∈T. U≠0 ⟶ A∩U≠0
  {
    fix x
    assume A:x∈⋃T
    then have ∀U∈T. x∈U ⟶ U∩A≠0 using AS by auto
    with assms A have x∈Closure(A,T) using topology0.inter_neigh_cl topology0_def
by auto
  }
  then have ⋃T⊆Closure(A,T) by auto
  with assms show Closure(A,T)=⋃T using topology0.Top_3_L11(1) topology0_def
by blast
qed
```

80.3 Relations between numerability properties and choice principles

It is known that some statements in topology aren't just derived from choice axioms, but also equivalent to them. Here is an example

The following are equivalent:

- Every topological space of second cardinality csucc(Q) is separable of cardinality csucc(Q).
- The axiom of Q choice.

In the article [4] there is a proof of this statement for Q= $\mathbb{N}$, with more equivalences.

If a topology is of second type of cardinal csucc(Q), then it is separable of the same cardinal. This result makes use of the axiom of choice for the cardinal Q on subsets of $\bigcup$T.

```
theorem Q_choice_imp_second_imp_separable:
  assumes T{is of second type of cardinal}csucc(Q)
    and {the axiom of} Q {choice holds for subsets} ⋃T
    and T{is a topology}
  shows T{is separable of cardinal}csucc(Q)
proof-
  from assms(1) have ∃B. (B {is a base for} T) ∧ (B ≺ csucc(Q)) us-
ing IsSecondOfCard_def by auto
  then obtain B where base:(B {is a base for} T) ∧ (B ≺ csucc(Q)) by
auto
  let N=λb∈B. b
  let B=B-{0}
  have B-{0}⊆B by auto
  with base have prec:B-{0}≺csucc(Q) using subset_imp_lepoll lesspoll_trans1
by blast
  from base have baseOpen:∀b∈B. Nb∈T using base_sets_open by auto
  from assms(2) have car:Card(Q) and reg:(∀ M N. (M ≲Q ∧  (∀t∈M. Nt≠0
∧ Nt⊆⋃T)) ⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)))
  using AxiomCardinalChoice_def by auto
  then have (B ≲Q ∧  (∀t∈B. Nt≠0 ∧ Nt⊆⋃T)) ⟶ (∃f. f:Pi(B,λt. Nt)
∧ (∀t∈B. ft∈Nt)) by blast
  with prec have (∀t∈B. Nt⊆⋃T) ⟶ (∃f. f:Pi(B,λt. Nt) ∧ (∀t∈B. ft∈Nt))
using Card_less_csucc_eq_le car by auto
  with baseOpen have ∃f. f:Pi(B,λt. Nt) ∧ (∀t∈B. ft∈Nt) by blast
  then obtain f where f:f:Pi(B,λt. Nt) and f2:∀t∈B. ft∈Nt by auto
  {
    fix U
    assume U∈T and U≠0
    then obtain b where A1:b∈B-{0} and b⊆U using Top_1_2_L1 base by
blast
    with f2 have fb∈U by auto
    with A1 have {fb. b∈B}∩U≠0 by auto
  }
  then have r:∀U∈T. U≠0 ⟶ {fb. b∈B}∩U≠0 by auto
  have {fb. b∈B}⊆⋃T using f2 baseOpen by auto
  moreover
  with r have Closure({fb. b∈B},T)=⋃T using dense_int_open assms(3)
by auto
  moreover
  have ffun:f:B→range(f) using f range_of_fun by auto
  then have f∈surj(B,range(f)) using fun_is_surj by auto
  then have des1:range(f)≲B using surj_fun_inv_2[of fBrange(f)Q] prec
Card_less_csucc_eq_le car
    Card_is_Ord by auto
  then have {fb. b∈B}⊆range(f) using apply_rangeI[OF ffun] by auto
  then have {fb. b∈B}≲range(f) using subset_imp_lepoll by auto
  with des1 have {fb. b∈B}≲B using lepoll_trans by blast
```

```
  with prec have {fb. b∈B}≺csucc(Q) using lesspoll_trans1 by auto
  ultimately show thesis using IsSeparableOfCard_def by auto
qed
```

The next theorem resolves that the axiom of Q choice for subsets of $\bigcup$T is necessary for second type spaces to be separable of the same cardinal csucc(Q).

```
theorem second_imp_separable_imp_Q_choice:
  assumes ∀T. (T{is a topology} ∧ (T{is of second type of cardinal}csucc(Q)))
⟶ (T{is separable of cardinal}csucc(Q))
  and Card(Q)
  shows {the axiom of} Q {choice holds}
proof-
  {
    fix N M
    assume AS:M ≲Q ∧  (∀t∈M. Nt≠0)

    then obtain h where inj:h∈inj(M,Q) using lepoll_def by auto
    then have bij:converse(h):bij(range(h),M) using inj_bij_range bij_converse_bij
by auto
    let T={(N(converse(h)i))×{i}. i∈range(h)}
    {
      fix j
      assume AS2:j∈range(h)
      from bij have converse(h):range(h)→M using bij_def inj_def by
auto
      with AS2 have converse(h)j∈M by simp
      with AS have N(converse(h)j)≠0 by auto
      then have (N(converse(h)j))×{j}≠0 by auto
    }
    then have noEmpty:0∉T by auto
    moreover
    {
      fix A B
      assume AS2:A∈TB∈TA∩B≠0
      then obtain j t where A_def:A=N(converse(h)j)×{j} and B_def:B=N(converse(h)t)×{t}
        and Range:j∈range(h) t∈range(h) by auto
      from AS2(3) obtain x where x∈A∩B by auto
      with A_def B_def have j=t by auto
      with A_def B_def have A=B by auto
    }
    then have (∀A∈T. ∀B∈T. A=B∨ A∩B=0) by auto
    ultimately
    have Part:T {is a partition of} ⋃T unfolding IsAPartition_def by
auto
    let τ=PTopology ⋃T T
    from Part have top:τ {is a topology} and base:T {is a base for}τ
      using Ptopology_is_a_topology by auto
    let f={⟨i,(N(converse(h)i))×{i}⟩. i∈range(h)}
```

```
    have f:range(h)→T using functionI[of f] Pi_def by auto
    then have f∈surj(range(h),T) unfolding surj_def using apply_equality
by auto
    moreover
    have range(h)⊆Q using inj unfolding inj_def range_def domain_def
Pi_def by auto
    ultimately have T≲ Q using surj_fun_inv[of frange(h)TQ] assms(2)
Card_is_Ord lepoll_trans
      subset_imp_lepoll by auto
    then have  T≺csucc(Q) using Card_less_csucc_eq_le assms(2) by auto
    with base have (τ{is of second type of cardinal}csucc(Q)) using IsSecondOfCard_def
by auto
    with top have τ{is separable of cardinal}csucc(Q) using assms(1)
by auto
    then obtain D where sub:D∈Pow(⋃τ) and clos:Closure(D,τ)=⋃τ and
cardd:D≺csucc(Q)
      using IsSeparableOfCard_def by auto

    then have D≲Q using Card_less_csucc_eq_le assms(2) by auto
    then obtain r where r:r∈inj(D,Q) using lepoll_def by auto
    then have bij2:converse(r):bij(range(r),D) using inj_bij_range bij_converse_bij
by auto
    then have surj2:converse(r):surj(range(r),D) using bij_def by auto
    let R=λi∈range(h). {j∈range(r). converse(r)j∈((N(converse(h)i))×{i})}
    {
      fix i
      assume AS:i∈range(h)
      then have T:(N(converse(h)i))×{i}∈T by auto
      then have P: (N(converse(h)i))×{i}∈τ using base unfolding IsAbaseFor_def
by blast
      with top sub clos have ∀U∈τ. U≠0 ⟶ D∩U≠0 using dense_int_open
by auto
      with P have (N(converse(h)i))×{i}≠0 ⟶ D∩(N(converse(h)i))×{i}≠0
by auto
      with T noEmpty have D∩(N(converse(h)i))×{i}≠0 by auto
      then obtain x where x∈D and px:x∈(N(converse(h)i))×{i} by auto
      with surj2 obtain j where j∈range(r) and converse(r)j=x unfold-
ing surj_def by blast
      with px have j∈{j∈range(r). converse(r)j∈((N(converse(h)i))×{i})}
by auto
      then have Ri≠0 using beta_if[of range(h) _ i] AS by auto
    }
    then have nonE:∀i∈range(h). Ri≠0 by auto
    {
      fix i j
      assume i:i∈range(h) and j:j∈Ri
      from j i have converse(r)j∈((N(converse(h)i))×{i}) using beta_if
by auto
    }
```

```
    then have pp:∀i∈range(h). ∀j∈Ri. converse(r)j∈((N(converse(h)i))×{i})
by auto
    let E={⟨m,fst(converse(r)(μ j. j∈R(hm)))⟩. m∈M}
    have ff:function(E) unfolding function_def by auto
    moreover

    {
      fix m
      assume M:m∈M
      with inj have hm:hm∈range(h) using apply_rangeI inj_def by auto
      {
        fix j
        assume j∈R(hm)
        with hm have j∈range(r) using beta_if by auto
        from r have r:surj(D,range(r)) using fun_is_surj inj_def by auto
        with ‹j∈range(r)› obtain d where d∈D and rd=j using surj_def
by auto
        then have j∈Q using r inj_def by auto
      }
      then have subcar:R(hm)⊆Q by blast
      from nonE hm obtain ee where P:ee∈R(hm) by blast
      with subcar have ee∈Q by auto
      then have Ord(ee) using assms(2) Card_is_Ord Ord_in_Ord by auto
      with P have (μ j. j∈R(hm))∈R(hm) using LeastI[where i=ee and P=λj.
j∈R(hm)]
      by auto
      with pp hm have converse(r)(μ j. j∈R(hm))∈((N(converse(h)(hm)))×{(hm)})
by auto
      then have converse(r)(μ j. j∈R(hm))∈((N(m))×{(hm)}) using left_inverse[OF
inj M]
        by simp
      then have fst(converse(r)(μ j. j∈R(hm)))∈(N(m)) by auto
      }
    ultimately have thesis1:∀m∈M. Em∈(N(m)) using function_apply_equality
by auto
    {
      fix e
      assume e∈E
      then obtain m where m∈M and e=⟨m,Em⟩ using function_apply_equality
ff by auto
      with thesis1 have e∈Sigma(M,λt. Nt) by auto
    }
    then have E∈Pow(Sigma(M,λt. Nt)) by auto
    with ff have E∈Pi(M,λm. Nm) using Pi_iff by auto
    then have (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) using thesis1 by
auto
  }
  then show thesis using AxiomCardinalChoiceGen_def assms(2) by auto
qed
```

Here is the equivalence from the two previous results.

```
theorem Q_choice_eq_secon_imp_sepa:
  assumes Card(Q)
  shows (∀T. (T{is a topology} ∧ (T{is of second type of cardinal}csucc(Q)))
⟶ (T{is separable of cardinal}csucc(Q)))
    ⟷({the axiom of} Q {choice holds})
  using Q_choice_imp_second_imp_separable choice_subset_imp_choice
  using second_imp_separable_imp_Q_choice assms by auto
```

Given a base injective with a set, then we can find a base whose elements are indexed by that set.

```
lemma base_to_indexed_base:
  assumes B ≲Q B {is a base for}T
  shows ∃N. {Ni. i∈Q}{is a base for}T
proof-
  from assms obtain f where f_def:f∈inj(B,Q) unfolding lepoll_def by
auto
  let ff={⟨b,fb⟩. b∈B}
  have domain(ff)=B by auto
  moreover
  have relation(ff) unfolding relation_def by auto
  moreover
  have function(ff) unfolding function_def by auto
  ultimately
  have fun:ff:B→range(ff) using function_imp_Pi[of ff] by auto
  then have injj:ff∈inj(B,range(ff)) unfolding inj_def
  proof
    {
      fix w x
      assume AS:w∈Bx∈B{⟨b, f  b⟩ . b ∈ B}  w = {⟨b, f  b⟩ . b ∈ B}  x
      then have fw=fx using apply_equality[OF _ fun] by auto
      then have w=x using f_def inj_def AS(1,2) by auto
    }
    then show ∀w∈B. ∀x∈B. {⟨b, f  b⟩ . b ∈ B}  w = {⟨b, f  b⟩ . b ∈
B}  x ⟶ w = x by auto
  qed
  then have bij:ff∈bij(B,range(ff)) using inj_bij_range by auto
  from fun have range(ff)={fb. b∈B} by auto
  with f_def have ran:range(ff)⊆Q using inj_def by auto
  let N={⟨i,(if i∈range(ff) then converse(ff)i else 0)⟩. i∈Q}
  have FN:function(N) unfolding function_def by auto
  have B ⊆{Ni. i∈Q}
  proof
    fix t
    assume a:t∈B
    from bij have rr:ff:B→range(ff) unfolding bij_def inj_def by auto
    have ig:fft=ft using a apply_equality[OF _ rr] by auto
    have r:fft∈range(ff) using apply_type[OF rr a].
    from ig have t:fft∈Q using apply_type[OF _ a] f_def unfolding inj_def
```

```
by auto
    with r have N(fft)=converse(ff)(fft) using function_apply_equality[OF
_ FN] by auto
    then have N(fft)=t using left_inverse[OF injj a] by auto
    then have t=N(fft) by auto
    then have ∃i∈Q. t=Ni using t(1) by auto
    then show t∈{Ni. i∈Q} by simp
  qed
  moreover
  have ∀r∈{Ni. i∈Q}-B. r=0
  proof
    fix r
    assume r∈{Ni. i∈Q}-B
    then obtain j where R:j∈Qr=Njr∉B by auto
    {
      assume AS:j∈range(ff)
      with R(1) have Nj=converse(ff)j using function_apply_equality[OF
_ FN] by auto
      then have Nj∈B using  apply_funtype[OF inj_is_fun[OF bij_is_inj[OF
bij_converse_bij[OF bij]]] AS]
      by auto
      then have False using R(3,2) by auto
    }
    then have j∉range(ff) by auto
    then show r=0 using function_apply_equality[OF _ FN] R(1,2) by auto
  qed
  ultimately have {Ni. i∈Q}=B∨{Ni. i∈Q}=B ∪{0} by blast
  moreover
  have (B ∪{0})-{0}=B-{0} by blast
  then have (B ∪{0})-{0} {is a base for}T using base_no_0[of BT] assms(2)
by auto
  then have B ∪{0} {is a base for}T using base_no_0[of B ∪{0}T] by auto
  ultimately
  have {Ni. i∈Q}{is a base for}T using assms(2) by auto
  then show thesis by auto
qed
```

80.4 Relation between numerability and compactness

If the axiom of Q choice holds, then any topology of second type of cardinal csucc(Q) is compact of cardinal csucc(Q)

```
theorem compact_of_cardinal_Q:
  assumes {the axiom of} Q {choice holds for subsets} (Pow(Q))
    T{is of second type of cardinal}csucc(Q)
    T{is a topology}
  shows ((⋃T){is compact of cardinal}csucc(Q){in}T)
proof-
  from assms(1) have CC:Card(Q) and reg:⋀ M N. (M ≲Q ∧ (∀t∈M. Nt≠0∧Nt⊆Pow(Q)))
⟶ (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) using
```

```
  AxiomCardinalChoice_def by auto
  from assms(2) obtain R where R≲QR{is a base for}T unfolding IsSecondOfCard_def
using Card_less_csucc_eq_le CC by auto
  with base_to_indexed_base obtain N where base:{Ni. i∈Q}{is a base for}T
by blast
  {
    fix M
    assume A:⋃T⊆⋃MM∈Pow(T)
    let α=λU∈M. {i∈Q. N(i)⊆U}
    have inj:α∈inj(M,Pow(Q)) unfolding inj_def
    proof
    {
      show (λU∈M. {i ∈ Q . N  i ⊆ U}) ∈ M → Pow(Q) using lam_type[of
MλU. {i ∈ Q . N(i) ⊆ U}%t. Pow(Q)] by auto
      {
        fix w x
        assume AS:w∈Mx∈M{i ∈ Q . N(i) ⊆ w} = {i ∈ Q . N(i) ⊆ x}
        from AS(1,2) A(2) have w∈Tx∈T by auto
        then have w=Interior(w,T)x=Interior(x,T) using assms(3) topology0.Top_2_L3[of
T]
          topology0_def[of T] by auto
        then have UN:w=(⋃{B∈{N(i). i∈Q}. B⊆w})x=(⋃{B∈{N(i). i∈Q}.
B⊆x})
          using interior_set_base_topology assms(3) base by auto
        {
          fix b
          assume b∈w
          then have b∈⋃{B∈{N(i). i∈Q}. B⊆w} using UN(1) by auto
          then obtain S where S:S∈{N(i). i∈Q} b∈S S⊆w by blast
          then obtain j where j:j∈QS=N(j) by auto
          then have j∈{i ∈ Q . N(i) ⊆ w} using S(3) by auto
          then have N(j)⊆xb∈N(j)j∈Q using S(2) AS(3) j by auto
          then have b∈(⋃{B∈{N(i). i∈Q}. B⊆x}) by auto
          then have b∈x using UN(2) by auto
        }
        moreover
        {
          fix b
          assume b∈x
          then have b∈⋃{B∈{N(i). i∈Q}. B⊆x} using UN(2) by auto
          then obtain S where S:S∈{N(i). i∈Q} b∈S S⊆x by blast
          then obtain j where j:j∈QS=N(j) by auto
          then have j∈{i ∈ Q . N(i) ⊆ x} using S(3) by auto
          then have j∈{i ∈ Q . N(i) ⊆ w} using AS(3) by auto
          then have N(j)⊆wb∈N(j)j∈Q using S(2) j(2) by auto
          then have b∈(⋃{B∈{N(i). i∈Q}. B⊆w}) by auto
          then have b∈w using UN(2) by auto
        }
        ultimately have w=x by auto
```

```
      }
      then show ∀w∈M. ∀x∈M. (λU∈M. {i ∈ Q . N  i ⊆ U})  w = (λU∈M.
{i ∈ Q . N  i ⊆ U})  x ⟶ w = x by auto
    }
    qed
    let X=λi∈Q. {αU. U∈{V∈M. N(i)⊆V}}
    let M={i∈Q. Xi≠0}
    have subMQ:M⊆Q by auto
    then have ddd:M ≲Q using subset_imp_lepoll by auto
    then have M ≲Q∀i∈M. Xi≠0∀i∈M. Xi⊆Pow(Q) by auto
    then have M ≲Q∀i∈M. Xi≠0∀i∈M. Xi ≲ Pow(Q) using subset_imp_lepoll
by auto
    then have (∃f. f:Pi(M,λt. Xt) ∧ (∀t∈M. ft∈Xt)) using reg[of MX]
by auto
    then obtain f where f:f:Pi(M,λt. Xt)(!!t. t∈M ⟹ ft∈Xt) by auto
    {
      fix m
      assume S:m∈M
      from f(2) S obtain YY where YY:(YY∈M) (fm=αYY) by auto
      then have Y:(YY∈M)∧(fm=αYY) by auto
      moreover
      {
        fix U
        assume U∈M∧(fm=αU)
        then have U=YY using inj inj_def YY by auto
      }
      then have r:⋀x. x∈M∧(fm=αx) ⟹ x=YY by blast
      have ∃!YY. YY∈M ∧ fm=αYY using ex1I[of %Y. Y∈M∧ fm=αY,OF Y r]
by auto
    }
    then have ex1YY:∀m∈M. ∃!YY. YY∈M ∧ fm=αYY by auto
    let YYm={⟨m,(THE YY. YY∈M ∧ fm=αYY)⟩. m∈M}
    have aux:⋀m. m∈M ⟹ YYmm=(THE YY. YY∈M ∧ fm=αYY) unfolding apply_def
by auto
    have ree:∀m∈M. (YYmm)∈M ∧ fm=α(YYmm)
    proof
      fix m
      assume C:m∈M
      then have ∃!YY. YY∈M ∧ fm=αYY using ex1YY by auto
      then have (THE YY. YY∈M ∧ fm=αYY)∈M∧fm=α(THE YY. YY∈M ∧ fm=αYY)
        using theI[of %Y. Y∈M∧ fm=αY] by blast
      then show (YYmm)∈M ∧ fm=α(YYmm) apply (simp only: aux[OF C]) done
    qed
    have tt:⋀m. m∈M ⟹ N(m)⊆YYmm
    proof-
      fix m
      assume D:m∈M
      then have QQ:m∈Q by auto
      from D have t:(YYmm)∈M ∧ fm=α(YYmm) using ree by blast
```

```
      then have fm=α(YYmm) by blast
      then have (α(YYmm))∈(λi∈Q. {αU. U∈{V∈M. N(i)⊆V}})m using f(2)[OF
D]
        by auto
      then have (α(YYmm))∈{αU. U∈{V∈M. N(m)⊆V}} using QQ by auto
      then obtain U where U∈{V∈M. N(m)⊆V}α(YYmm)=αU by auto
      then have r:U∈MN(m)⊆Uα(YYmm)=αU(YYmm)∈M using t by auto
      then have YYmm=U using  inj_apply_equality[OF inj] by blast
      then show N(m)⊆YYmm using r by auto
    qed
    then have (⋃m∈M. N(m))⊆(⋃m∈M. YYmm)
    proof-
      {
        fix s
        assume s∈(⋃m∈M. N(m))
        then obtain t where r:t∈Ms∈N(t) by auto
        then have s∈YYmt using tt[OF r(1)] by blast
        then have s∈(⋃m∈M. YYmm) using r(1) by blast
      }
      then show thesis by blast
    qed
    moreover
    {
      fix x
      assume AT:x∈⋃T
      with A obtain U where BB:U∈MU∈Tx∈U by auto
      then obtain j where BC:j∈Q N(j)⊆Ux∈N(j) using point_open_base_neigh[OF
base,of Ux] by auto
      then have Xj≠0 using BB(1) by auto
      then have j∈M using BC(1) by auto
      then have x∈(⋃m∈M. N(m)) using BC(3) by auto
    }
    then have ⋃T⊆(⋃m∈M. N(m)) by blast
    ultimately have covers:⋃T⊆(⋃m∈M. YYmm) using subset_trans[of ⋃T(⋃m∈M.
N(m))(⋃m∈M. YYmm)]
      by auto
    have relation(YYm) unfolding relation_def by auto
    moreover
    have f:function(YYm) unfolding function_def by auto
    moreover
    have d:domain(YYm)=M by auto
    moreover
    have r:range(YYm)=YYmM by auto
    ultimately
    have fun:YYm:M→YYmM using function_imp_Pi[of YYm] by auto
    have YYm∈surj(M,YYmM) using fun_is_surj[OF fun] r by auto
    with surj_fun_inv[OF this subMQ Card_is_Ord[OF CC]]
    have YYmM ≲ M by auto
    with ddd have Rw:YYmM ≲Q using lepoll_trans by blast
```

```
    {
      fix m assume m∈M
      then have ⟨m,YYmm⟩∈YYm using function_apply_Pair[OF f] d by blast
      then have YYmm∈YYmM by auto}
      then have l1:{YYmm. m∈M}⊆YYmM by blast
      {
        fix t assume t∈YYmM
        then have ∃x∈M. ⟨x,t⟩∈YYm unfolding image_def by auto
        then obtain r where S:r∈M⟨r,t⟩∈YYm by auto
        have YYmr=t using apply_equality[OF S(2) fun] by auto
        with S(1) have t∈{YYmm. m∈M} by auto
      }
      with l1 have {YYmm. m∈M}=YYmM by blast
      with Rw have {YYmm. m∈M} ≲Q by auto
      with covers have {YYmm. m∈M}∈Pow(M)∧⋃T⊆⋃{YYmm. m∈M}∧{YYmm. m∈M}
≺csucc(Q) using ree
        Card_less_csucc_eq_le[OF CC] by blast
      then have ∃N∈Pow(M). ⋃T⊆⋃N∧N≺csucc(Q) by auto
    }
  then have ∀M∈Pow(T). ⋃T ⊆ ⋃M ⟶ (∃N∈Pow(M). ⋃T ⊆ ⋃N ∧ N ≺ csucc(Q))
by auto
  then show thesis using IsCompactOfCard_def Card_csucc CC Card_is_Ord
by auto
qed
```

In the following proof, we have chosen an infinite cardinal to be able to apply the equation `Q × Q ≈ Q`. For finite cardinals; both, the assumption and the axiom of choice, are always true.

```
theorem second_imp_compact_imp_Q_choice_PowQ:
  assumes ∀T. (T{is a topology} ∧ (T{is of second type of cardinal}csucc(Q)))
⟶ ((⋃T){is compact of cardinal}csucc(Q){in}T)
  and InfCard(Q)
  shows {the axiom of} Q {choice holds for subsets} (Pow(Q))
proof-
  {
    fix N M
    assume AS:M ≲Q ∧  (∀t∈M. Nt≠0 ∧ Nt⊆Pow(Q))
    then obtain h where h∈inj(M,Q) using lepoll_def by auto

    have discTop:Pow(Q×M) {is a topology} using Pow_is_top by auto
    {
      fix A
      assume AS:A∈Pow(Q×M)
      have A=⋃{{i}. i∈A} by auto
      with AS have ∃T∈Pow({{i}. i∈Q×M}). A=⋃T by auto
      then have A∈{⋃U. U∈Pow({{i}. i∈Q×M})} by auto
    }
    moreover
    {
```

fix A
assume AS:A∈{⋃U. U∈Pow({{i}. i∈Q×M})}
then have A∈Pow(Q×M) by auto
}
ultimately
have base:{{x}. x∈Q×M} {is a base for} Pow(Q×M) unfolding IsAbaseFor_def
by blast
let f={⟨i,{i}⟩. i∈Q×M}
have fff:f∈Q×M→{{i}. i∈Q×M} using Pi_def function_def by auto
then have f∈inj(Q×M,{{i}. i∈Q×M}) unfolding inj_def using apply_equality
by auto
then have f∈bij(Q×M,{{i}. i∈Q×M}) unfolding bij_def surj_def using fff
apply_equality fff by auto
then have Q×M≈{{i}. i∈Q×M} using eqpoll_def by auto
then have {{i}. i∈Q×M}≈Q×M using eqpoll_sym by auto
then have {{i}. i∈Q×M}≲Q×M using eqpoll_imp_lepoll by auto
then have {{i}. i∈Q×M}≲Q×Q using AS prod_lepoll_mono[of QQMQ] lepoll_refl[of
Q]
lepoll_trans by blast
then have {{i}. i∈Q×M}≲Q using InfCard_square_eqpoll assms(2) lepoll_eq_trans
by auto
then have {{i}. i∈Q×M}≺csucc(Q) using Card_less_csucc_eq_le assms(2)
InfCard_is_Card by auto
then have Pow(Q×M) {is of second type of cardinal} csucc(Q) using
IsSecondOfCard_def base by auto
then have comp:(Q×M) {is compact of cardinal}csucc(Q){in}Pow(Q×M)
using discTop assms(1) by auto
{
fix W
assume W∈Pow(Q×M)
then have T:W{is closed in} Pow(Q×M) and (Q×M)∩W=W using IsClosed_def
by auto
with compact_closed[OF comp T] have (W {is compact of cardinal}csucc(Q){in}Pow(Q×M))
by auto
}
then have subCompact:∀W∈Pow(Q×M). (W {is compact of cardinal}csucc(Q){in}Pow(Q×M))
by auto
let cub=⋃{{(U)×{t}. U∈Nt}. t∈M}
from AS have (⋃cub)∈Pow((Q)×M) by auto
with subCompact have Ncomp:((⋃cub) {is compact of cardinal}csucc(Q){in}Pow(Q×M))
by auto
have cond:(cub)∈Pow(Pow(Q×M))∧ ⋃cub⊆⋃cub using AS by auto
have ∃S∈Pow(cub). (⋃cub) ⊆ ⋃S ∧ S ≺ csucc(Q)
proof-
{
have ((⋃cub) {is compact of cardinal}csucc(Q){in}Pow(Q×M)) using Ncomp by auto
then have ∀M∈Pow(Pow(Q×M)). ⋃cub ⊆ ⋃M ⟶ (∃Na∈Pow(M). ⋃cub

```
⊆ ⋃Na ∧ Na ≺ csucc(Q))
          unfolding IsCompactOfCard_def by auto
        with cond have ∃S∈Pow(cub). ⋃cub ⊆ ⋃S ∧ S ≺ csucc(Q) by auto
      }
      then show thesis by auto
    qed
    then have ttt:∃S∈Pow(cub). (⋃cub) ⊆ ⋃S ∧ S ≲ Q using Card_less_csucc_eq_le
assms(2) InfCard_is_Card by auto
    then obtain S where S_def:S∈Pow(cub)(⋃cub) ⊆ ⋃S S ≲ Q by auto
    {
      fix t
      assume AA:t∈MNt≠{0}
      from AA(1) AS have Nt≠0 by auto
      with AA(2) obtain U where G:U∈Nt and notEm:U≠0 by blast
      then have U×{t}∈cub using AA by auto
      then have U×{t}⊆⋃cub by auto
      with G notEm AA have ∃s. ⟨s,t⟩∈⋃cub by auto
    }
    then have ∀t∈M. (Nt≠{0})⟶ (∃s. ⟨s,t⟩∈⋃cub) by auto
    then have A:∀t∈M. (Nt≠{0})⟶ (∃s. ⟨s,t⟩∈⋃S) using S_def(2) by
blast
    from S_def(1) have B:∀f∈S. ∃t∈M. ∃U∈Nt. f=U×{t} by blast
    from A B have ∀t∈M. (Nt≠{0})⟶ (∃U∈Nt. U×{t}∈S) by blast
    then have noEmp:∀t∈M. (Nt≠{0})⟶ (S∩({U×{t}. U∈Nt})≠0) by auto
    from S_def(3) obtain r where r:r:inj(S,Q) using lepoll_def by auto
    then have bij2:converse(r):bij(range(r),S) using inj_bij_range bij_converse_bij
by auto
    then have surj2:converse(r):surj(range(r),S) using bij_def by auto
    let R=λt∈M. {j∈range(r). converse(r)j∈({U×{t}. U∈Nt})}
    {
      fix t
      assume AA:t∈MNt≠{0}
      then have (S∩({U×{t}. U∈Nt})≠0) using noEmp by auto
      then obtain s where ss:s∈Ss∈{U×{t}. U∈Nt} by blast
      then obtain j where converse(r)j=s j∈range(r) using surj2 unfold-
ing surj_def by blast
      then have j∈{j∈range(r). converse(r)j∈({U×{t}. U∈Nt})} using ss
by auto
      then have Rt≠0 using beta_if AA by auto
    }
    then have nonE:∀t∈M. Nt≠{0}⟶Rt≠0 by auto
    {
      fix t j
      assume t∈Mj∈Rt
      then have converse(r)j∈{U×{t}. U∈Nt} using beta_if by auto
      }
    then have pp:∀t∈M. ∀j∈Rt. converse(r)j∈{U×{t}. U∈Nt} by auto
    have reg:∀t U V. U×{t}=V×{t}⟶U=V
    proof-
```

```
      {
        fix t U V
        assume AA:U×{t}=V×{t}
        {
          fix v
          assume v∈V
          then have ⟨v,t⟩∈V ×{t} by auto
          then have ⟨v,t⟩∈U ×{t} using AA by auto
          then have v∈U by auto
        }
        then have V⊆U by auto
        moreover
        {
          fix u
          assume u∈U
          then have ⟨u,t⟩∈U ×{t} by auto
          then have ⟨u,t⟩∈V ×{t} using AA by auto
          then have u∈V by auto
        }
        then have U⊆V by auto
        ultimately have U=V by auto
      }
      then show thesis by auto
    qed

    let E={⟨t,if Nt={0} then 0 else (THE U. converse(r)(μ j. j∈Rt)=U×{t})⟩.
t∈M}
    have ff:function(E) unfolding function_def by auto
    moreover
    {
      fix t
      assume pm:t∈M
       { assume nonEE:Nt≠{0}
      {
        fix j
        assume j∈Rt
        with pm(1) have j∈range(r) using beta_if by auto
        from r have r:surj(S,range(r)) using fun_is_surj inj_def by auto
        with ‹j∈range(r)› obtain d where d∈S and rd=j using surj_def
by auto
        then have j∈Q using r inj_def by auto
        }
      then have sub:Rt⊆Q by blast
      from nonE pm nonEE obtain ee where P:ee∈Rt by blast
      with sub have ee∈Q by auto
      then have Ord(ee) using assms(2) Card_is_Ord Ord_in_Ord InfCard_is_Card
by blast
      with P have (μ j. j∈Rt)∈Rt using LeastI[where i=ee and P=λj.
j∈Rt] by auto
```

```
      with pp pm have converse(r)(μ j. j∈Rt)∈{U×{t}. U∈Nt} by auto
      then obtain W where converse(r)(μ j. j∈Rt)=W×{t} and s:W∈Nt by
auto
      then have (THE U. converse(r)(μ j. j∈Rt)=U×{t})=W using reg by
auto
      with s have (THE U. converse(r)(μ j. j∈Rt)=U×{t})∈Nt by auto
    }
    then have (if Nt={0} then 0 else (THE U. converse(r)(μ j. j∈Rt)=U×{t}))∈Nt
by auto
    }
    ultimately have thesis1:∀t∈M. Et∈Nt using function_apply_equality
by auto
    {
      fix e
      assume e∈E
      then obtain m where m∈M and e=⟨m,Em⟩ using function_apply_equality
ff by auto
      with thesis1 have e∈Sigma(M,λt. Nt) by auto
    }
    then have E∈Pow(Sigma(M,λt. Nt)) by auto
    with ff have E∈Pi(M,λm. Nm) using Pi_iff by auto
    then have (∃f. f:Pi(M,λt. Nt) ∧ (∀t∈M. ft∈Nt)) using thesis1 by
auto}
    then show thesis using AxiomCardinalChoice_def assms(2) InfCard_is_Card
by auto
qed
```

The two previous results, state the following equivalence:

```
theorem Q_choice_Pow_eq_secon_imp_comp:
  assumes InfCard(Q)
  shows  (∀T. (T{is a topology} ∧ (T{is of second type of cardinal}csucc(Q)))
⟶ ((⋃T){is compact of cardinal}csucc(Q){in}T))
    ⟷({the axiom of} Q {choice holds for subsets} (Pow(Q)))
    using second_imp_compact_imp_Q_choice_PowQ compact_of_cardinal_Q assms
by auto
```

In the next result we will prove that if the space $(\kappa, Pow(\kappa))$, for κ an infinite cardinal, is compact of its successor cardinal; then all topologycal spaces which are of second type of the successor cardinal of κ are also compact of that cardinal.

```
theorem Q_csuccQ_comp_eq_Q_choice_Pow:
  assumes InfCard(Q) (Q){is compact of cardinal}csucc(Q){in}Pow(Q)
  shows ∀T. (T{is a topology} ∧ (T{is of second type of cardinal}csucc(Q)))
⟶ ((⋃T){is compact of cardinal}csucc(Q){in}T)
proof
  fix T
  {
    assume top:T {is a topology} and sec:T{is of second type of cardinal}csucc(Q)
```

```
    from assms have Card(csucc(Q)) Card(Q) using InfCard_is_Card Card_is_Ord
Card_csucc by auto
    moreover
    have ⋃T⊆⋃T by auto
    moreover
    {
      fix M
      assume MT:M∈Pow(T) and cover:⋃T⊆⋃M
      from sec obtain B where B {is a base for} T B≺csucc(Q) using IsSecondOfCard_def
by auto
      with ‹Card(Q)› obtain N where base:{Ni. i∈Q}{is a base for}T us-
ing Card_less_csucc_eq_le
        base_to_indexed_base by blast
      let S={⟨u,{i∈Q. Ni⊆u}⟩. u∈M}
      have function(S) unfolding function_def by auto
      then have S:M→Pow(Q) using Pi_iff by auto
      then have S∈inj(M,Pow(Q)) unfolding inj_def
        proof
        {
          fix w x
          assume AS:w∈Mx∈M{⟨u, {i ∈ Q . N  i ⊆ u}⟩ . u ∈ M}  w = {⟨u,
{i ∈ Q . N  i ⊆ u}⟩ . u ∈ M}  x
          with ‹S:M→Pow(Q)› have ASS:{i ∈ Q . N  i ⊆ w}={i ∈ Q . N
i ⊆ x} using apply_equality by auto
          from AS(1,2) MT have w∈Tx∈T by auto
          then have w=Interior(w,T)x=Interior(x,T) using top topology0.Top_2_L3[of
T]
            topology0_def[of T] by auto
          then have UN:w=(⋃{B∈{N(i). i∈Q}. B⊆w})x=(⋃{B∈{N(i). i∈Q}.
B⊆x})
            using interior_set_base_topology top base by auto
          {
            fix b
            assume b∈w
            then have b∈⋃{B∈{N(i). i∈Q}. B⊆w} using UN(1) by auto
            then obtain S where S:S∈{N(i). i∈Q} b∈S S⊆w by blast
            then obtain j where j:j∈QS=N(j) by auto
            then have j∈{i ∈ Q . N(i) ⊆ w} using S(3) by auto
            then have N(j)⊆xb∈N(j)j∈Q using S(2) ASS j by auto
            then have b∈(⋃{B∈{N(i). i∈Q}. B⊆x}) by auto
            then have b∈x using UN(2) by auto
          }
          moreover
          {
            fix b
            assume b∈x
            then have b∈⋃{B∈{N(i). i∈Q}. B⊆x} using UN(2) by auto
            then obtain S where S:S∈{N(i). i∈Q} b∈S S⊆x by blast
            then obtain j where j:j∈QS=N(j) by auto
```

```
            then have j∈{i ∈ Q . N(i) ⊆ x} using S(3) by auto
            then have j∈{i ∈ Q . N(i) ⊆ w} using ASS by auto
            then have N(j)⊆wb∈N(j)j∈Q using S(2) j(2) by auto
            then have b∈(⋃{B∈{N(i). i∈Q}. B⊆w}) by auto
            then have b∈w using UN(2) by auto
          }
          ultimately have w=x by auto
        }
        then show ∀w∈M. ∀x∈M. {⟨u, {i ∈ Q . N  i ⊆ u}⟩ . u ∈ M}  w
= {⟨u, {i ∈ Q . N  i ⊆ u}⟩ . u ∈ M}  x ⟶ w = x by auto
      qed
      then have S∈bij(M,range(S)) using fun_is_surj unfolding bij_def
inj_def surj_def by force
      have range(S)⊆Pow(Q) by auto
      then have range(S)∈Pow(Pow(Q)) by auto
      moreover
      have (⋃(range(S))) {is closed in} Pow(Q) Q∩(⋃range(S))=(⋃range(S))
using IsClosed_def by auto
      from this(2) compact_closed[OF assms(2) this(1)] have (⋃range(S)){is
compact of cardinal}csucc(Q) {in}Pow(Q)
        by auto
      moreover
      have ⋃(range(S))⊆⋃(range(S)) by auto
      ultimately have ∃S∈Pow(range(S)). (⋃(range(S)))⊆⋃S ∧ S≺ csucc(Q)
using IsCompactOfCard_def by auto
      then obtain SS where SS_def:SS⊆range(S) (⋃(range(S)))⊆⋃SS SS≺
csucc(Q) by auto
      with ‹S∈bij(M,range(S))› have con:converse(S)∈bij(range(S),M) us-
ing bij_converse_bij by auto
      then have r1:restrict(converse(S),SS)∈bij(SS,converse(S)SS) us-
ing restrict_bij bij_def SS_def(1) by auto
      then have rr:converse(restrict(converse(S),SS))∈bij(converse(S)SS,SS)
using bij_converse_bij by auto
      {
        fix x
        assume x∈⋃T
        with cover have x∈⋃M by auto
        then obtain R where R∈M x∈R by auto
        with MT have R∈T x∈R by auto
        then have ∃V∈{Ni. i∈Q}. V⊆R ∧ x∈V using point_open_base_neigh
base by force
        then obtain j where j∈Q Nj⊆R and x_p:x∈Nj by auto
        with ‹R∈M› ‹S:M→Pow(Q)› ‹S∈bij(M,range(S))› have SR∈range(S)
∧ j∈SR using apply_equality
          bij_def inj_def by auto
        from exI[where P=λt. t∈range(S) ∧ j∈t, OF this] have ∃A∈range(S).
j∈A unfolding Bex_def
          by auto
        then have j∈(⋃(range(S))) by auto
```

```
        then have j∈⋃SS using SS_def(2) by blast
        then obtain SR where SR∈SS j∈SR by auto
        moreover
        have converse(restrict(converse(S),SS))∈surj(converse(S)SS,SS)
using rr bij_def by auto
        ultimately obtain RR where converse(restrict(converse(S),SS))RR=SR
and p:RR∈converse(S)SS unfolding surj_def by blast
        then have converse(converse(restrict(converse(S),SS)))(converse(restrict(converse(S
          by auto
        moreover
        have converse(restrict(converse(S),SS))∈inj(converse(S)SS,SS)
using rr unfolding bij_def by auto
        moreover
        ultimately have RR=converse(converse(restrict(converse(S),SS)))SR
using left_inverse[OF _ p]
          by force
        moreover
        with r1 have restrict(converse(S),SS)∈SS→converse(S)SS unfold-
ing bij_def inj_def by auto
        then have relation(restrict(converse(S),SS)) using Pi_def relation_def
by auto
        then have converse(converse(restrict(converse(S),SS)))=restrict(converse(S),SS)
using relation_converse_converse by auto
        ultimately have RR=restrict(converse(S),SS)SR by auto
        with ‹SR∈SS› have eq:RR=converse(S)SR unfolding restrict by auto
        then have converse(converse(S))RR=converse(converse(S))(converse(S)SR)
by auto
        moreover
        with ‹SR∈SS› have SR∈range(S) using SS_def(1) by auto
        from con left_inverse[OF _ this] have converse(converse(S))(converse(S)SR)=SR
unfolding bij_def
          by auto
        ultimately have converse(converse(S))RR=SR by auto
        then have SRR=SR using relation_converse_converse[of S] unfold-
ing relation_def by auto
        moreover
        have converse(S):range(S)→M using con bij_def inj_def by auto
        with ‹SR∈range(S)› have converse(S)SR∈M using apply_funtype
          by auto
        with eq have RR∈M by auto
        ultimately have SR={i∈Q. Ni⊆RR} using ‹S:M→Pow(Q)› apply_equality
by auto
        then have Nj⊆RR using ‹j∈SR› by auto
        with x_p have x∈RR by auto
        with p have x∈⋃(converse(S)SS) by auto
      }
      then have ⋃T⊆⋃(converse(S)SS) by blast
      moreover
      {
```

```
        from con have converse(S)SS={converse(S)R. R∈SS} using image_function[of
converse(S) SS]
          SS_def(1) unfolding range_def bij_def inj_def Pi_def by auto
        have {converse(S)R. R∈SS}⊆{converse(S)R. R∈range(S)} using SS_def(1)
by auto
        moreover
        have converse(S):range(S)→M using con unfolding bij_def inj_def
by auto
        then have {converse(S)R. R∈range(S)}⊆M using apply_funtype by
force
        ultimately
        have (converse(S)SS)⊆M by auto
      }
      then have converse(S)SS∈Pow(M) by auto
      moreover
      with rr have converse(S)SS≈SS using eqpoll_def by auto
      then have converse(S)SS≺csucc(Q) using SS_def(3) eq_lesspoll_trans
by auto
      ultimately
      have ∃N∈Pow(M). ⋃T⊆⋃N ∧ N≺csucc(Q) by auto
    }
    then have ∀M∈Pow(T). ⋃T⊆⋃M ⟶ (∃N∈Pow(M). ⋃T⊆⋃N ∧ N≺csucc(Q))
by auto
    ultimately have (⋃T){is compact of cardinal}csucc(Q){in}T unfold-
ing IsCompactOfCard_def
      by auto
  }
  then show (T {is a topology}) ∧ (T {is of second type of cardinal}csucc(Q))
⟶ ((⋃T){is compact of cardinal}csucc(Q) {in}T)
  by auto
qed

theorem Q_disc_is_second_card_csuccQ:
  assumes InfCard(Q)
  shows Pow(Q){is of second type of cardinal}csucc(Q)
proof-
  {
    fix A
    assume AS:A∈Pow(Q)
    have A=⋃{{i}. i∈A} by auto
    with AS have ∃T∈Pow({{i}. i∈Q}). A=⋃T by auto
    then have A∈{⋃U. U∈Pow({{i}. i∈Q})} by auto
  }
  moreover
  {
    fix A
    assume AS:A∈{⋃U. U∈Pow({{i}. i∈Q})}
    then have A∈Pow(Q) by auto
  }
```

ultimately
  have base:{{x}. x∈Q} {is a base for} Pow(Q) unfolding IsAbaseFor_def
by blast
  let f={⟨i,{i}⟩. i∈Q}
  have f∈Q→{{x}. x∈Q} unfolding Pi_def function_def by auto
  then have f∈inj(Q,{{x}. x∈Q}) unfolding inj_def using apply_equality
by auto
  moreover
  from ‹f∈Q→{{x}. x∈Q}› have f∈surj(Q,{{x}. x∈Q}) unfolding surj_def
using apply_equality
    by auto
  ultimately have f∈bij(Q,{{x}. x∈Q}) unfolding bij_def by auto
  then have Q≈{{x}. x∈Q} using eqpoll_def by auto
  then have {{x}. x∈Q}≈Q using eqpoll_sym by auto
  then have {{x}. x∈Q}≲Q using eqpoll_imp_lepoll by auto
  then have {{x}. x∈Q}≺csucc(Q) using Card_less_csucc_eq_le assms InfCard_is_Card
by auto
  with base show thesis using IsSecondOfCard_def by auto
qed

This previous results give us another equivalence of the axiom of Q choice that is apparently weaker (easier to check) to the previous one.

theorem Q_disc_comp_csuccQ_eq_Q_choice_csuccQ:
  assumes InfCard(Q)
  shows (Q{is compact of cardinal}csucc(Q){in}(Pow(Q))) ⟷ ({the axiom of}Q{choice holds for subsets}(Pow(Q)))
  proof
  assume Q{is compact of cardinal}csucc(Q) {in}Pow(Q)
  with assms show {the axiom of}Q{choice holds for subsets}(Pow(Q)) using Q_choice_Pow_eq_secon_imp_comp Q_csuccQ_comp_eq_Q_choice_Pow
    by auto
  next
  assume {the axiom of}Q{choice holds for subsets}(Pow(Q))
  with assms show Q{is compact of cardinal}csucc(Q){in}(Pow(Q)) using
Q_disc_is_second_card_csuccQ Q_choice_Pow_eq_secon_imp_comp Pow_is_top[of
Q]
    by force
qed

end

81 Topology 5

theory Topology_ZF_5 imports Topology_ZF_properties Topology_ZF_examples_1
Topology_ZF_4
begin

81.1 Some results for separation axioms

First we will give a global characterization of T_1-spaces; which is interesting because it involves the cardinal ℕ.

```
lemma (in topology0) T1_cocardinal_coarser:
  shows (T {is T₁}) ⟷ (CoFinite (⋃T))⊆T
proof
  {
    assume AS:T {is T₁}
    {
      fix x assume p:x∈⋃T
      {
        fix y assume y∈(⋃T)-{x}
        with AS p obtain U where U∈T y∈U x∉U using isT1_def by blast
        then have U∈T y∈U U⊆(⋃T)-{x} by auto
        then have ∃U∈T. y∈U ∧ U⊆(⋃T)-{x} by auto
      }
      then have ∀y∈(⋃T)-{x}. ∃U∈T. y∈U ∧ U⊆(⋃T)-{x} by auto
      then have ⋃T-{x}∈T using open_neigh_open by auto
      with p have {x} {is closed in}T using IsClosed_def by auto
    }
    then have pointCl:∀x∈⋃T. {x} {is closed in} T by auto
    {
      fix A
      assume AS2:A∈FinPow(⋃T)
      let p={⟨x,{x}⟩. x∈A}
      have p∈A→{{x}. x∈A} using Pi_def unfolding function_def by auto
      then have p:bij(A,{{x}. x∈A}) unfolding bij_def inj_def surj_def
using apply_equality
        by auto
      then have A≈{{x}. x∈A} unfolding eqpoll_def by auto
      with AS2 have Finite({{x}. x∈A}) unfolding FinPow_def using eqpoll_imp_Finite_iff
by auto
      then have {{x}. x∈A}∈FinPow({D ∈ Pow(⋃T) . D {is closed in} T})
using AS2 pointCl unfolding FinPow_def
      by (safe, blast+)
      then have (⋃{{x}. x∈A}) {is closed in} T using fin_union_cl_is_cl
by auto
      moreover
      have ⋃{{x}. x∈A}=A by auto
      ultimately have A {is closed in} T by simp
    }
    then have reg:∀A∈FinPow(⋃T). A {is closed in} T by auto
    {
      fix U
      assume AS2:U ∈ CoCardinal(⋃T,nat)
      then have U∈Pow(⋃T) U=0 ∨ ((⋃T)-U)≺nat using CoCardinal_def by
auto
      then have U∈Pow(⋃T) U=0 ∨ Finite(⋃T-U) using lesspoll_nat_is_Finite
```

```
by auto
      then have U∈Pow(⋃T) U∈T∨(⋃T-U) {is closed in} T using empty_open
topSpaceAssum
        reg unfolding FinPow_def by auto
      then have U∈Pow(⋃T) U∈T∨(⋃T-(⋃T-U))∈T using IsClosed_def by
auto
      moreover
      then have (⋃T-(⋃T-U))=U by blast
      ultimately have U∈T by auto
    }
    then show (CoFinite (⋃T))⊆T using Cofinite_def by auto
  }
  {
    assume (CoFinite (⋃T))⊆T
    then have AS:CoCardinal(⋃T,nat) ⊆ T using Cofinite_def by auto
    {
      fix x y
      assume AS2:x∈⋃T y∈⋃Tx≠y
      have Finite({y}) by auto
      then obtain n where {y}≈n n∈nat using Finite_def by auto
      then have {y}≺nat using n_lesspoll_nat eq_lesspoll_trans by auto
      then have {y} {is closed in} CoCardinal(⋃T,nat) using closed_sets_cocardinal
        AS2(2) by auto
      then have (⋃T)-{y}∈CoCardinal(⋃T,nat) using union_cocardinal
IsClosed_def by auto
      with AS have (⋃T)-{y}∈T by auto
      moreover
      with AS2(1,3) have x∈((⋃T)-{y}) ∧ y∉((⋃T)-{y}) by auto
      ultimately have ∃V∈T. x∈V∧y∉V by(safe,auto)
    }
    then show T {is T₁} using isT1_def by auto
  }
qed
```

In the previous proof, it is obvious that we don't need to check if ever cofinite set is open. It is enough to check if every singleton is closed.

```
corollary(in topology0) T1_iff_singleton_closed:
  shows (T {is T₁}) ⟷ (∀x∈⋃T. {x}{is closed in}T)
proof
  assume AS:T {is T₁}
  {
    fix x assume p:x∈⋃T
    {
      fix y assume y∈(⋃T)-{x}
      with AS p obtain U where U∈T y∈U x∉U using isT1_def by blast
      then have U∈T y∈U U⊆(⋃T)-{x} by auto
      then have ∃U∈T. y∈U ∧ U⊆(⋃T)-{x} by auto
    }
    then have ∀y∈(⋃T)-{x}. ∃U∈T. y∈U ∧ U⊆(⋃T)-{x} by auto
```

```
    then have ⋃T-{x}∈T using open_neigh_open by auto
    with p have {x} {is closed in}T using IsClosed_def by auto
  }
  then show pointCl:∀x∈⋃T. {x} {is closed in} T by auto
next
  assume pointCl:∀x∈⋃T. {x} {is closed in} T
  {
    fix A
    assume AS2:A∈FinPow(⋃T)
    let p={⟨x,{x}⟩. x∈A}
    have p∈A→{{x}. x∈A} using Pi_def unfolding function_def by auto
    then have p:bij(A,{{x}. x∈A}) unfolding bij_def inj_def surj_def
using apply_equality
      by auto
    then have A≈{{x}. x∈A} unfolding eqpoll_def by auto
    with AS2 have Finite({{x}. x∈A}) unfolding FinPow_def using eqpoll_imp_Finite_iff
by auto
    then have {{x}. x∈A}∈FinPow({D ∈ Pow(⋃T) . D {is closed in} T})
using AS2 pointCl unfolding FinPow_def
    by (safe, blast+)
    then have (⋃{{x}. x∈A}) {is closed in} T using fin_union_cl_is_cl
by auto
    moreover
    have ⋃{{x}. x∈A}=A by auto
    ultimately have A {is closed in} T by simp
  }
  then have reg:∀A∈FinPow(⋃T). A {is closed in} T by auto
  {
    fix U
    assume AS2:U∈CoCardinal(⋃T,nat)
    then have U∈Pow(⋃T) U=0 ∨ ((⋃T)-U)≺nat using CoCardinal_def by
auto
    then have U∈Pow(⋃T) U=0 ∨ Finite(⋃T-U) using lesspoll_nat_is_Finite
by auto
    then have U∈Pow(⋃T) U∈T∨(⋃T-U) {is closed in} T using empty_open
topSpaceAssum
      reg unfolding FinPow_def by auto
    then have U∈Pow(⋃T) U∈T∨(⋃T-(⋃T-U))∈T using IsClosed_def by auto
    moreover
    then have (⋃T-(⋃T-U))=U by blast
    ultimately have U∈T by auto
  }
  then have (CoFinite (⋃T))⊆T using Cofinite_def by auto
  then show T {is T₁} using T1_cocardinal_coarser by auto
qed
```

Secondly, let's show that the `CoCardinal X Q` topologies for different sets Q are all ordered as the partial order of sets. (The order is linear when considering only cardinals)

```
lemma order_cocardinal_top:
  fixes X
  assumes Q1≲Q2
  shows CoCardinal(X,Q1) ⊆ CoCardinal(X,Q2)
proof
  fix x
  assume x ∈ CoCardinal(X,Q1)
  then have x∈Pow(X) x=0∨(X-x)≺Q1 using CoCardinal_def by auto
  with assms have x∈Pow(X) x=0∨(X-x)≺Q2 using lesspoll_trans2 by auto
  then show x∈CoCardinal(X,Q2) using CoCardinal_def by auto
qed

corollary cocardinal_is_T1:
  fixes X K
  assumes InfCard(K)
  shows CoCardinal(X,K) {is T₁}
proof-
  have nat≤K using InfCard_def assms by auto
  then have nat⊆K using le_imp_subset by auto
  then have nat≲K K≠0using subset_imp_lepoll by auto
  then have CoCardinal(X,nat) ⊆ CoCardinal(X,K) ⋃CoCardinal(X,K)=X us-
ing order_cocardinal_top
    union_cocardinal by auto
  then show thesis using topology0.T1_cocardinal_coarser topology0_CoCardinal
assms Cofinite_def
    by auto
qed
```

In T_2-spaces, filters and nets have at most one limit point.

```
lemma (in topology0) T2_imp_unique_limit_filter:
  assumes T {is T₂} 𝔉 {is a filter on}⋃T 𝔉 →_F x 𝔉 →_F y
  shows x=y
proof-
  {
    assume x≠y
    from assms(3,4) have x∈⋃T y∈⋃T using FilterConverges_def assms(2)
      by auto
    with ‹x≠y› have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0 using assms(1) isT2_def
by auto
    then obtain U V where x∈U y∈V U∩V=0 U∈T V∈T by auto
    then have U∈{A∈Pow(⋃T). x∈Interior(A,T)} V∈{A∈Pow(⋃T). y∈Interior(A,T)}
using Top_2_L3 by auto
    then have U∈𝔉 V∈𝔉 using FilterConverges_def assms(2) assms(3,4)
      by auto
    then have U∩V∈𝔉 using IsFilter_def assms(2) by auto
    with ‹U∩V=0› have 0∈𝔉 by auto
    then have False using IsFilter_def assms(2) by auto
  }
  then show thesis by auto
```

```
qed

lemma (in topology0) T2_imp_unique_limit_net:
  assumes T {is T₂} N {is a net on}⋃T N →N x N →N y
  shows x=y
proof-
  have (Filter N..(⋃T)) {is a filter on} (⋃T) (Filter N..(⋃T)) →F
x (Filter N..(⋃T)) →F y
    using filter_of_net_is_filter(1) net_conver_filter_of_net_conver assms(2)
    assms(3,4) by auto
  with assms(1) show thesis using T2_imp_unique_limit_filter by auto
qed
```

In fact, T_2-spaces are characterized by this property. For this proof we build a filter containing the union of two filters.

```
lemma (in topology0) unique_limit_filter_imp_T2:
  assumes ∀x∈⋃T. ∀y∈⋃T. ∀𝔉. ((𝔉 {is a filter on}⋃T) ∧ (𝔉 →F x)
∧ (𝔉 →F y)) ⟶ x=y
  shows T {is T₂}
proof-
  {
    fix x y
    assume x∈⋃T y∈⋃T x≠y
    {
      assume ∀U∈T. ∀V∈T. (x∈U ∧ y∈V) ⟶ U∩V≠0
      let Ux={A∈Pow(⋃T). x∈int(A)}
      let Uy={A∈Pow(⋃T). y∈int(A)}
      let FF=Ux ∪ Uy ∪ {A∩B. ⟨A,B⟩∈Ux × Uy}
      have sat:FF {satisfies the filter base condition}
      proof-
        {
          fix A B
          assume A∈FF B∈FF
          {
            assume A∈Ux
            {
              assume B∈Ux
              with ‹x∈⋃T› ‹A∈Ux› have A∩B∈Ux using neigh_filter(1)
IsFilter_def by auto
              then have A∩B∈FF by auto
            }
            moreover
            {
              assume B∈Uy
              with ‹A∈Ux› have A∩B∈FF by auto
            }
            moreover
            {
              assume B∈{A∩B. ⟨A,B⟩∈Ux × Uy}
```

```
          then obtain AA BB where B=AA∩BB AA∈Ux BB∈Uy by auto
          with ‹x∈⋃T› ‹A∈Ux› have A∩B=(A∩AA)∩BB A∩AA∈Ux using neigh_filter(1)
IsFilter_def by auto
          with ‹BB∈Uy› have A∩B∈{A∩B. ⟨A,B⟩∈Ux × Uy} by auto
          then have A∩B∈FF by auto
        }
        ultimately have A∩B∈FF using ‹B∈FF› by auto
      }
      moreover
      {
        assume A∈Uy
        {
          assume B∈Uy
          with ‹y∈⋃T› ‹A∈Uy› have A∩B∈Uy using neigh_filter(1)
IsFilter_def by auto
          then have A∩B∈FF by auto
        }
        moreover
        {
          assume B∈Ux
          with ‹A∈Uy› have B∩A∈FF by auto
          moreover have A∩B=B∩A by auto
          ultimately have A∩B∈FF by auto
        }
        moreover
        {
          assume B∈{A∩B. ⟨A,B⟩∈Ux × Uy}
          then obtain AA BB where B=AA∩BB AA∈Ux BB∈Uy by auto
          with ‹y∈⋃T› ‹A∈Uy› have A∩B=AA∩(A∩BB) A∩BB∈Uy using neigh_filter(1)
IsFilter_def by auto
          with ‹AA∈Ux› have A∩B∈{A∩B. ⟨A,B⟩∈Ux × Uy} by auto
          then have A∩B∈FF by auto
        }
        ultimately have A∩B∈FF using ‹B∈FF› by auto
      }
      moreover
      {
        assume A∈{A∩B. ⟨A,B⟩∈Ux × Uy}
        then obtain AA BB where A=AA∩BB AA∈Ux BB∈Uy by auto
        {
          assume B∈Uy
          with ‹BB∈Uy› ‹y∈⋃T› have B∩BB∈Uy using neigh_filter(1)
IsFilter_def by auto
          moreover from ‹A=AA∩BB› have A∩B=AA∩(B∩BB) by auto
          ultimately have A∩B∈FF using ‹AA∈Ux› ‹B∩BB∈Uy› by auto
        }
        moreover
        {
          assume B∈Ux
```

```
              with ‹AA∈Ux› ‹x∈⋃T› have B∩AA∈Ux using neigh_filter(1)
IsFilter_def by auto
              moreover from ‹A=AA∩BB› have A∩B=(B∩AA)∩BB by auto
              ultimately have A∩B∈FF using ‹B∩AA∈Ux› ‹BB∈Uy› by auto
            }
            moreover
            {
              assume B∈{A∩B. ⟨A,B⟩∈Ux × Uy}
              then obtain AA2 BB2 where B=AA2∩BB2 AA2∈Ux BB2∈Uy by auto
              from ‹B=AA2∩BB2› ‹A=AA∩BB› have A∩B=(AA∩AA2)∩(BB∩BB2)
by auto
              moreover
              from ‹AA∈Ux›‹AA2∈Ux›‹x∈⋃T› have AA∩AA2∈Ux using neigh_filter(1)
IsFilter_def by auto
              moreover
              from ‹BB∈Uy›‹BB2∈Uy›‹y∈⋃T› have BB∩BB2∈Uy using neigh_filter(1)
IsFilter_def by auto
              ultimately have A∩B∈FF by auto
            }
            ultimately have A∩B∈FF using ‹B∈FF› by auto
          }
          ultimately have A∩B∈FF using ‹A∈FF› by auto
          then have ∃D∈FF. D⊆A∩B unfolding Bex_def by auto
        }
        then have ∀A∈FF. ∀B∈FF. ∃D∈FF. D⊆A∩B by force
        moreover
        have ⋃T∈Ux using ‹x∈⋃T› neigh_filter(1) IsFilter_def by auto
        then have FF≠0 by auto
        moreover
        {
          assume 0∈FF
          moreover
          have 0∉Ux using ‹x∈⋃T› neigh_filter(1) IsFilter_def by auto
          moreover
          have 0∉Uy using ‹y∈⋃T› neigh_filter(1) IsFilter_def by auto
          ultimately have 0∈{A∩B. ⟨A,B⟩∈Ux × Uy} by auto
          then obtain A B where 0=A∩B A∈UxB∈Uy by auto
          then have x∈int(A)y∈int(B) by auto
          moreover
          with ‹0=A∩B› have int(A)∩int(B)=0 using Top_2_L1 by auto
          moreover
          have int(A)∈Tint(B)∈T using Top_2_L2 by auto
          ultimately have False using ‹∀U∈T. ∀V∈T. x∈U∧y∈V ⟶ U∩V≠0›
by auto
        }
        then have 0∉FF by auto
        ultimately show thesis using SatisfiesFilterBase_def by auto
      qed
      moreover
```

```
      have FF⊆Pow(⋃T) by auto
      ultimately have bas:FF {is a base filter} {A∈Pow(⋃T). ∃D∈FF. D⊆A}
⋃{A∈Pow(⋃T). ∃D∈FF. D⊆A}=⋃T
        using base_unique_filter_set2[of FF] by auto
      then have fil:{A∈Pow(⋃T). ∃D∈FF. D⊆A} {is a filter on} ⋃T us-
ing basic_filter sat by auto
      have ∀U∈Pow(⋃T). x∈int(U) ⟶ (∃D∈FF. D⊆U) by auto
      then have {A∈Pow(⋃T). ∃D∈FF. D⊆A} →F x using convergence_filter_base2[OF
fil bas(1) _ ‹x∈⋃T›] by auto
      moreover
      then have ∀U∈Pow(⋃T). y∈int(U) ⟶ (∃D∈FF. D⊆U) by auto
      then have {A∈Pow(⋃T). ∃D∈FF. D⊆A} →F y using convergence_filter_base2[OF
fil bas(1) _ ‹y∈⋃T›] by auto
      ultimately have x=y using assms fil ‹x∈⋃T›‹y∈⋃T› by blast
      with ‹x≠y› have False by auto
    }
    then have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0 by blast
  }
  then show thesis using isT2_def by auto
qed

lemma (in topology0) unique_limit_net_imp_T2:
  assumes ∀x∈⋃T. ∀y∈⋃T. ∀N. ((N {is a net on}⋃T) ∧ (N →N x) ∧ (N
→N y)) ⟶ x=y
  shows T {is T2}
proof-
  {
    fix x y 𝔉
    assume x∈⋃T y∈⋃T𝔉 {is a filter on}⋃T𝔉 →F x𝔉 →F y
    then have (Net(𝔉)) {is a net on} ⋃T(Net 𝔉) →N x(Net 𝔉) →N y
      using filter_conver_net_of_filter_conver net_of_filter_is_net by
auto
    with  ‹x∈⋃T› ‹y∈⋃T› have x=y using assms by blast
  }
  then have ∀x∈⋃T. ∀y∈⋃T. ∀𝔉. ((𝔉 {is a filter on}⋃T) ∧ (𝔉 →F x)
∧ (𝔉 →F y)) ⟶ x=y by auto
  then show thesis using unique_limit_filter_imp_T2 by auto
qed
```

This results make easy to check if a space is T_2.

The topology which comes from a filter as in `𝔉 {is a filter on} ⋃𝔉 ⟹ (𝔉 ∪ cons(∅, ∅)) {is a topology}` is not T_2 generally. We will see in this file later on, that the exceptions are a consequence of the spectrum.

```
corollary filter_T2_imp_card1:
  assumes (𝔉∪{0}) {is T2} 𝔉 {is a filter on} ⋃𝔉 x∈⋃𝔉
  shows ⋃𝔉={x}
proof-
  {
```

```
    fix y assume y∈⋃𝔉
    then have 𝔉 →F y {in} (𝔉∪{0}) using lim_filter_top_of_filter assms(2)
by auto
    moreover
    have 𝔉 →F x {in} (𝔉∪{0}) using lim_filter_top_of_filter assms(2,3)
by auto
    moreover
    have ⋃𝔉=⋃(𝔉∪{0}) by auto
    ultimately
    have y=x using topology0.T2_imp_unique_limit_filter[OF topology0_filter[OF
assms(2)] assms(1)] assms(2)
      by auto
  }
  then have ⋃𝔉⊆{x} by auto
  with assms(3) show thesis by auto
qed
```

There are more separation axioms that just T_0, T_1 or T_2

```
definition
  IsNormal (_{is normal} 90)
  where T{is normal} ≡ ∀A. A{is closed in}T ⟶ (∀B. B{is closed in}T
∧ A∩B=0 ⟶
  (∃U∈T. ∃V∈T. A⊆U∧B⊆V∧U∩V=0))

definition
  isT4 (_{is T4} 90)
  where T{is T4} ≡ (T{is T1}) ∧ (T{is normal})

lemma (in topology0) T4_is_T3:
  assumes T{is T4} shows T{is T3}
proof-
  from assms have nor:T{is normal} using isT4_def by auto
  from assms have T{is T1} using isT4_def by auto
  then have Cofinite (⋃T)⊆T using T1_cocardinal_coarser by auto
  {
    fix A
    assume AS:A{is closed in}T
    {
      fix x
      assume x∈⋃T-A
      have Finite({x}) by auto
      then obtain n where {x}≈n n∈nat unfolding Finite_def by auto
      then have {x}≲n n∈nat using eqpoll_imp_lepoll by auto
      then have {x}≺nat using n_lesspoll_nat lesspoll_trans1 by auto
      with ‹x∈⋃T-A› have {x} {is closed in} (Cofinite (⋃T)) using Cofinite_def

        closed_sets_cocardinal by auto
      then have ⋃T-{x}∈Cofinite(⋃T) unfolding IsClosed_def using union_cocardinal
Cofinite_def
```

```
        by auto
      with ‹Cofinite (⋃T)⊆T› have ⋃T-{x}∈T by auto
      with ‹x∈⋃T-A› have {x}{is closed in}T A∩{x}=0 using IsClosed_def
by auto
      with nor AS have ∃U∈T. ∃V∈T. A⊆U∧{x}⊆V∧U∩V=0 unfolding IsNormal_def
by blast
      then have ∃U∈T. ∃V∈T. A⊆U∧ x∈V∧U∩V=0 by auto
    }
    then have ∀x∈⋃T-A. ∃U∈T. ∃V∈T. A⊆U∧ x∈V∧U∩V=0 by auto
  }
  then have T{is regular} using IsRegular_def by blast
  with ‹T{is T₁}› show thesis unfolding isT3_def using T1_is_T0 by simp
qed
```

Regularity can be rewritten in terms of existence of certain neighboorhoods.

```
lemma (in topology0) regular_imp_exist_clos_neig:
  assumes T{is regular} and U∈T and x∈U
  shows ∃V∈T. x∈V ∧ cl(V)⊆U
proof-
  from assms(2) have (⋃T-U){is closed in}T using Top_3_L9 by auto more-
over
  from assms(2,3) have x∈⋃T by auto moreover
  note assms(1,3) ultimately obtain A B where A∈T and B∈T and A∩B=0
and (⋃T-U)⊆A and x∈B
    unfolding IsRegular_def by blast
  from ‹A∩B=0› ‹B∈T› have B⊆⋃T-A by auto
  with ‹A∈T› have cl(B)⊆⋃T-A using Top_3_L9 Top_3_L13 by auto
  moreover from ‹(⋃T-U)⊆A› assms(3) have ⋃T-A⊆U by auto
  moreover note ‹x∈B› ‹B∈T›
  ultimately have B∈T ∧ x∈B ∧ cl(B)⊆U by auto
  then show thesis by auto
qed

lemma (in topology0) exist_clos_neig_imp_regular:
  assumes ∀x∈⋃T. ∀U∈T. x∈U ⟶ (∃V∈T. x∈V∧ cl(V)⊆U)
  shows T{is regular}
proof-
  {
    fix F
    assume F{is closed in}T
    {
      fix x assume x∈⋃T-F
      with ‹F{is closed in}T› have x∈⋃T ⋃T-F∈T F⊆⋃T unfolding IsClosed_def
by auto
      with assms ‹x∈⋃T-F› have ∃V∈T. x∈V ∧ cl(V)⊆⋃T-F by auto
      then obtain V where V∈T x∈V cl(V)⊆⋃T-F by auto
      from ‹cl(V)⊆⋃T-F› ‹F⊆⋃T› have F⊆⋃T-cl(V) by auto
      moreover from ‹V∈T› have ⋃T-(⋃T-V)=V by auto
      then have cl(V)=⋃T-int(⋃T-V) using Top_3_L11(2)[of ⋃T-V] by
```

```
auto
      ultimately have F⊆int(⋃T-V) by auto moreover
      have int(⋃T-V)⊆⋃T-V using Top_2_L1 by auto
      then have V∩(int(⋃T-V))=0 by auto moreover
      note ‹x∈V›‹V∈T› ultimately
      have V∈T int(⋃T-V)∈T F⊆int(⋃T-V) ∧ x∈V ∧ (int(⋃T-V))∩V=0 us-
ing Top_2_L2
        by auto
      then have ∃U∈T. ∃V∈T. F⊆U ∧ x∈V ∧ U∩V=0 by auto
    }
    then have ∀x∈⋃T-F. ∃U∈T. ∃V∈T. F⊆U ∧ x∈V ∧ U∩V=0 by auto
  }
  then show thesis using IsRegular_def by blast
qed

lemma (in topology0) regular_eq:
  shows T{is regular} ⟷ (∀x∈⋃T. ∀U∈T. x∈U ⟶ (∃V∈T. x∈V∧ cl(V)⊆U))
  using regular_imp_exist_clos_neig exist_clos_neig_imp_regular by force
```

A Hausdorff space separates compact spaces from points.

```
theorem (in topology0) T2_compact_point:
  assumes T{is T₂} A{is compact in}T x∈⋃T x∉A
  shows ∃U∈T. ∃V∈T. A⊆U ∧ x∈V ∧ U∩V=0
proof-
  {
    assume A=0
    then have A⊆0∧x∈⋃T∧(0∩(⋃T)=0) using assms(3) by auto
    then have thesis using empty_open topSpaceAssum unfolding IsATopology_def
by auto
  }
  moreover
  {
    assume noEmpty:A≠0
    let U={⟨U,V⟩∈T×T. x∈U∧U∩V=0}
    {
      fix y assume y∈A
      with ‹x∉A› assms(4) have x≠y by auto
      moreover from ‹y∈A› have x∈⋃Ty∈⋃T using assms(2,3) unfolding
IsCompact_def by auto
      ultimately obtain U V where U∈TV∈TU∩V=0x∈Uy∈V using assms(1) un-
folding isT2_def by blast
      then have ∃⟨U,V⟩∈U. y∈V by auto
    }
    then have ∀y∈A. ∃⟨U,V⟩∈U. y∈V by auto
    then have A⊆⋃{snd(B). B∈U} by auto
    moreover have {snd(B). B∈U}∈Pow(T) by auto
    ultimately have ∃N∈FinPow({snd(B). B∈U}). A⊆⋃N using assms(2) un-
folding IsCompact_def by auto
    then obtain N where ss:N∈FinPow({snd(B). B∈U}) A⊆⋃N by auto
```

```
    with ‹{snd(B). B∈U}∈Pow(T)› have A⊆⋃N N∈Pow(T) unfolding FinPow_def
by auto
    then have NN:A⊆⋃N ⋃N∈T using topSpaceAssum unfolding IsATopology_def
by auto
    from ss have Finite(N)N⊆{snd(B). B∈U} unfolding FinPow_def by auto
    then obtain n where n∈nat N≈n unfolding Finite_def by auto
    then have N≲n using eqpoll_imp_lepoll by auto
    from noEmpty ‹A⊆⋃N› have NnoEmpty:N≠0 by auto
    let QQ={⟨n,{fst(B). B∈{A∈U. snd(A)=n}}⟩. n∈N}
    have QQPi:QQ:N→{{fst(B). B∈{A∈U. snd(A)=n}}. n∈N} unfolding Pi_def
function_def domain_def by auto
    {
      fix n assume n∈N
      with ‹N⊆{snd(B). B∈U}› obtain B where n=snd(B) B∈U by auto
      then have fst(B)∈{fst(B). B∈{A∈U. snd(A)=n}} by auto
      then have {fst(B). B∈{A∈U. snd(A)=n}}≠0 by auto moreover
      from ‹n∈N› have ⟨n,{fst(B). B∈{A∈U. snd(A)=n}}⟩∈QQ by auto
      with QQPi have QQn={fst(B). B∈{A∈U. snd(A)=n}} using apply_equality
by auto
      ultimately have QQn≠0 by auto
    }
    then have ∀n∈N. QQn≠0 by auto
    with ‹n∈nat› ‹N≲n› have ∃f. f∈Pi(N,λt. QQt) ∧ (∀t∈N. ft∈QQt) us-
ing finite_choice unfolding AxiomCardinalChoiceGen_def
      by auto
    then obtain f where fPI:f∈Pi(N,λt. QQt) (∀t∈N. ft∈QQt) by auto
    from fPI(1) NnoEmpty have range(f)≠0 unfolding Pi_def range_def domain_def
converse_def by (safe,blast)
    {
      fix t assume t∈N
      then have ft∈QQt using fPI(2) by auto
      with ‹t∈N› have ft∈⋃(QQN) QQt⊆⋃(QQN) using func_imagedef QQPi
by auto
    }
    then have reg:∀t∈N. ft∈⋃(QQN)  ∀t∈N. QQt⊆⋃(QQN) by auto
    {
      fix tt assume tt∈f
      with fPI(1) have tt∈Sigma(N, ()(QQ)) unfolding Pi_def by auto
      then have tt∈(⋃xa∈N. ⋃y∈QQxa. {⟨xa,y⟩}) unfolding Sigma_def by
auto
      then obtain xa y where xa∈N y∈QQxa tt=⟨xa,y⟩ by auto
      with reg(2) have y∈⋃(QQN) by blast
      with ‹tt=⟨xa,y⟩› ‹xa∈N› have tt∈(⋃xa∈N. ⋃y∈⋃(QQN). {⟨xa,y⟩})
by auto
      then have tt∈N×(⋃(QQN)) unfolding Sigma_def by auto
    }
    then have ffun:f:N→⋃(QQN)  using fPI(1) unfolding Pi_def by auto
    then have f∈surj(N,range(f)) using fun_is_surj by auto
    with ‹N≲n› ‹n∈nat› have range(f)≲N using surj_fun_inv_2 nat_into_Ord
```

```
by auto
    with ‹N≲n› have range(f)≲n using lepoll_trans by blast
    with ‹n∈nat› have Finite(range(f)) using n_lesspoll_nat lesspoll_nat_is_Finite
lesspoll_trans1 by auto
    moreover from ffun have rr:range(f)⊆⋃(QQN) unfolding Pi_def by
auto
    then have range(f)⊆T by auto
    ultimately have range(f)∈FinPow(T) unfolding FinPow_def by auto
    then have ⋂range(f)∈T using fin_inter_open_open ‹range(f)≠0› by
auto moreover
    {
      fix S assume S∈range(f)
      with rr have S∈⋃(QQN) by blast
      then have ∃B∈(QQN). S ∈ B using Union_iff by auto
      then obtain B where B∈(QQN) S∈B by auto
      then have ∃rr∈N. ⟨rr,B⟩∈QQ unfolding image_def by auto
      then have ∃rr∈N. B={fst(B). B∈{A∈U. snd(A)=rr}} by auto
      with ‹S∈B› obtain rr where ⟨S,rr⟩∈U by auto
      then have x∈S by auto
    }
    then have x∈⋂range(f) using ‹range(f)≠0› by auto moreover
    {
      fix y assume y∈(⋃N)∩(⋂range(f))
      then have reg:(∀S∈range(f). y∈S)∧(∃t∈N. y∈t) by auto
      then obtain t where t∈N y∈t by auto
      then have ⟨t, {fst(B). B∈{A∈U. snd(A)=t}}⟩∈QQ by auto
      then have ft∈range(f) using apply_rangeI ffun by auto
      with reg have yft:y∈ft by auto
      with ‹t∈N› fPI(2) have ft∈QQt by auto
      with ‹t∈N› have ft∈{fst(B). B∈{A∈U. snd(A)=t}} using apply_equality
QQPi by auto
      then have ⟨ft,t⟩∈U by auto
      then have ft∩t=0 by auto
      with ‹y∈t› yft have False by auto
    }
    then have (⋃N)∩(⋂range(f))=0 by blast moreover
    note NN
    ultimately have thesis by auto
  }
  ultimately show thesis by auto
qed
```

A Hausdorff space separates compact spaces from other compact spaces.

```
theorem (in topology0) T2_compact_compact:
  assumes T{is T₂} A{is compact in}T B{is compact in}T A∩B=0
  shows ∃U∈T. ∃V∈T. A⊆U ∧ B⊆V ∧ U∩V=0
proof-
 {
    assume B=0
```

```
    then have A⊆⋃T∧B⊆0∧((⋃T)∩0=0) using assms(2) unfolding IsCompact_def
by auto moreover
    have 0∈T using empty_open topSpaceAssum by auto moreover
    have ⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto
ultimately
    have thesis by auto
  }
  moreover
  {
    assume noEmpty:B≠0
    let U={⟨U,V⟩∈T×T. A⊆U ∧ U∩V=0}
    {
      fix y assume y∈B
      then have y∈⋃T using assms(3) unfolding IsCompact_def by auto
      with ‹y∈B› have ∃U∈T. ∃V∈T. A⊆U ∧ y∈V ∧ U∩V=0 using T2_compact_point
assms(1,2,4) by auto
      then have ∃⟨U,V⟩∈U. y∈V by auto
    }
    then have ∀y∈B. ∃⟨U,V⟩∈U. y∈V by auto
    then have B⊆⋃{snd(B). B∈U} by auto
    moreover have {snd(B). B∈U}∈Pow(T) by auto
    ultimately have ∃N∈FinPow({snd(B). B∈U}). B⊆⋃N using assms(3) un-
folding IsCompact_def by auto
    then obtain N where ss:N∈FinPow({snd(B). B∈U}) B⊆⋃N by auto
    with ‹{snd(B). B∈U}∈Pow(T)› have B⊆⋃N N∈Pow(T) unfolding FinPow_def
by auto
    then have NN:B⊆⋃N ⋃N∈T using topSpaceAssum unfolding IsATopology_def
by auto
    from ss have Finite(N)N⊆{snd(B). B∈U} unfolding FinPow_def by auto
    then obtain n where n∈nat N≈n unfolding Finite_def by auto
    then have N≲n using eqpoll_imp_lepoll by auto
    from noEmpty ‹B⊆⋃N› have NnoEmpty:N≠0 by auto
    let QQ={⟨n,{fst(B). B∈{A∈U. snd(A)=n}}⟩. n∈N}
    have QQPi:QQ:N→{{fst(B). B∈{A∈U. snd(A)=n}}. n∈N} unfolding Pi_def
function_def domain_def by auto
    {
      fix n assume n∈N
      with ‹N⊆{snd(B). B∈U}› obtain B where n=snd(B) B∈U by auto
      then have fst(B)∈{fst(B). B∈{A∈U. snd(A)=n}} by auto
      then have {fst(B). B∈{A∈U. snd(A)=n}}≠0 by auto moreover
      from ‹n∈N› have ⟨n,{fst(B). B∈{A∈U. snd(A)=n}}⟩∈QQ by auto
      with QQPi have QQn={fst(B). B∈{A∈U. snd(A)=n}} using apply_equality
by auto
      ultimately have QQn≠0 by auto
    }
    then have ∀n∈N. QQn≠0 by auto
    with ‹n∈nat› ‹N≲n› have ∃f. f∈Pi(N,λt. QQt) ∧ (∀t∈N. ft∈QQt) us-
ing finite_choice unfolding AxiomCardinalChoiceGen_def
      by auto
```

```
    then obtain f where fPI:f∈Pi(N,λt. QQt) (∀t∈N. ft∈QQt) by auto
    from fPI(1) NnoEmpty have range(f)≠0 unfolding Pi_def range_def domain_def
converse_def by (safe,blast)
    {
      fix t assume t∈N
      then have ft∈QQt using fPI(2) by auto
      with ‹t∈N› have ft∈⋃(QQN) QQt⊆⋃(QQN) using func_imagedef QQPi
by auto
    }
    then have reg:∀t∈N. ft∈⋃(QQN)  ∀t∈N. QQt⊆⋃(QQN) by auto
    {
      fix tt assume tt∈f
      with fPI(1) have tt∈Sigma(N, ()(QQ)) unfolding Pi_def by auto
      then have tt∈(⋃xa∈N. ⋃y∈QQxa. {⟨xa,y⟩}) unfolding Sigma_def by
auto
      then obtain xa y where xa∈N y∈QQxa tt=⟨xa,y⟩ by auto
      with reg(2) have y∈⋃(QQN) by blast
      with ‹tt=⟨xa,y⟩› ‹xa∈N› have tt∈(⋃xa∈N. ⋃y∈⋃(QQN). {⟨xa,y⟩})
by auto
      then have tt∈N×(⋃(QQN)) unfolding Sigma_def by auto
    }
    then have ffun:f:N→⋃(QQN)  using fPI(1) unfolding Pi_def by auto
    then have f∈surj(N,range(f)) using fun_is_surj by auto
    with ‹N≲n› ‹n∈nat› have range(f)≲N using surj_fun_inv_2 nat_into_Ord
by auto
    with ‹N≲n› have range(f)≲n using lepoll_trans by blast
    with ‹n∈nat› have Finite(range(f)) using n_lesspoll_nat lesspoll_nat_is_Finite
lesspoll_trans1 by auto
    moreover from ffun have rr:range(f)⊆⋃(QQN) unfolding Pi_def by
auto
    then have range(f)⊆T by auto
    ultimately have range(f)∈FinPow(T) unfolding FinPow_def by auto
    then have ⋂range(f)∈T using fin_inter_open_open ‹range(f)≠0› by
auto moreover
    {
      fix S assume S∈range(f)
      with rr have S∈⋃(QQN) by blast
      then have ∃B∈(QQN). S ∈ B using Union_iff by auto
      then obtain B where B∈(QQN) S∈B by auto
      then have ∃rr∈N. ⟨rr,B⟩∈QQ unfolding image_def by auto
      then have ∃rr∈N. B={fst(B). B∈{A∈U. snd(A)=rr}} by auto
      with ‹S∈B› obtain rr where ⟨S,rr⟩∈U by auto
      then have A⊆S by auto
    }
    then have A⊆⋂range(f) using ‹range(f)≠0› by auto moreover
    {
      fix y assume y∈(⋃N)∩(⋂range(f))
      then have reg:(∀S∈range(f). y∈S)∧(∃t∈N. y∈t) by auto
      then obtain t where t∈N y∈t by auto
```

```
      then have ⟨t, {fst(B). B∈{A∈U. snd(A)=t}}⟩∈QQ by auto
      then have ft∈range(f) using apply_rangeI ffun by auto
      with reg have yft:y∈ft by auto
      with ‹t∈N› fPI(2) have ft∈QQt by auto
      with ‹t∈N› have ft∈{fst(B). B∈{A∈U. snd(A)=t}} using apply_equality
QQPi by auto
      then have ⟨ft,t⟩∈U by auto
      then have ft∩t=0 by auto
      with ‹y∈t› yft have False by auto
    }
    then have (⋂range(f))∩(⋃N)=0 by blast moreover
    note NN
    ultimately have thesis by auto
  }
  ultimately show thesis by auto
qed
```

A compact Hausdorff space is normal.

```
corollary (in topology0) T2_compact_is_normal:
  assumes T{is T₂} (⋃T){is compact in}T
  shows T{is normal} unfolding IsNormal_def
proof-
  from assms(2) have car_nat:(⋃T){is compact of cardinal}nat{in}T us-
ing Compact_is_card_nat by auto
  {
    fix A B assume A{is closed in}T B{is closed in}TA∩B=0
    then have com:((⋃T)∩A){is compact of cardinal}nat{in}T ((⋃T)∩B){is
compact of cardinal}nat{in}T using compact_closed[OF car_nat]
      by auto
    from ‹A{is closed in}T›‹B{is closed in}T› have (⋃T)∩A=A(⋃T)∩B=B
unfolding IsClosed_def by auto
    with com have A{is compact of cardinal}nat{in}T B{is compact of cardinal}nat{in}T
by auto
    then have A{is compact in}TB{is compact in}T using Compact_is_card_nat
by auto
    with ‹A∩B=0› have ∃U∈T. ∃V∈T. A⊆U ∧ B⊆V ∧ U∩V=0 using T2_compact_compact
assms(1) by auto
  }
  then show  ∀A. A {is closed in} T ⟶ (∀B. B {is closed in} T ∧ A
∩ B = 0 ⟶ (∃U∈T. ∃V∈T. A ⊆ U ∧ B ⊆ V ∧ U ∩ V = 0))
    by auto
qed
```

81.2 Hereditability

A topological property is hereditary if whenever a space has it, every subspace also has it.

```
definition IsHer (_{is hereditary} 90)
```

```
  where P {is hereditary} ≡ ∀T. T{is a topology} ∧ P(T) ⟶ (∀A∈Pow(⋃T).
P(T{restricted to}A))

lemma subspace_of_subspace:
  assumes A⊆BB⊆⋃T
  shows T{restricted to}A=(T{restricted to}B){restricted to}A
proof
  from assms have S:∀S∈T. A∩(B∩S)=A∩S by auto
  then show T {restricted to} A ⊆ T {restricted to} B {restricted to}
A unfolding RestrictedTo_def
    by auto
  from S show T {restricted to} B {restricted to} A ⊆ T {restricted
to} A unfolding RestrictedTo_def
    by auto
qed
```

The separation properties T_0, T_1, T_2 y T_3 are hereditary.

```
theorem regular_here:
  assumes T{is regular} A∈Pow(⋃T) shows (T{restricted to}A){is regular}
proof-
  {
    fix C
    assume A:C{is closed in}(T{restricted to}A)
    {fix y assume y∈⋃(T{restricted to}A)y∉C
    with A have (⋃(T{restricted to}A))-C∈(T{restricted to}A)C⊆⋃(T{restricted
to}A) y∈⋃(T{restricted to}A)y∉C unfolding IsClosed_def
      by auto
    moreover
    with assms(2) have ⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
    ultimately have A-C∈T{restricted to}A y∈Ay∉CC∈Pow(A) by auto
    then obtain S where S∈T A∩S=A-C y∈Ay∉C unfolding RestrictedTo_def
by auto
    then have y∈A-CA∩S=A-C by auto
    with ‹C∈Pow(A)› have y∈A∩SC=A-A∩S by auto
    then have y∈S C=A-S by auto
    with assms(2) have y∈S C⊆⋃T-S by auto
    moreover
    from ‹S∈T› have ⋃T-(⋃T-S)=S by auto
    moreover
    with ‹S∈T› have (⋃T-S) {is closed in}T using IsClosed_def by auto
    ultimately have y∈⋃T-(⋃T-S) (⋃T-S) {is closed in}T by auto
    with assms(1) have ∀y∈⋃T-(⋃T-S). ∃U∈T. ∃V∈T. (⋃T-S)⊆U∧y∈V∧U∩V=0
unfolding IsRegular_def by auto
    with ‹y∈⋃T-(⋃T-S)› have ∃U∈T. ∃V∈T. (⋃T-S)⊆U∧y∈V∧U∩V=0 by auto
    then obtain U V where U∈TV∈T ⋃T-S⊆Uy∈VU∩V=0 by auto
    then have A∩U∈(T{restricted to}A)A∩V∈(T{restricted to}A) C⊆Uy∈V(A∩U)∩(A∩V)=0
      unfolding RestrictedTo_def  using ‹C⊆⋃T-S› by auto
    moreover
```

```
    with ‹C∈Pow(A)›‹y∈A› have C⊆A∩Uy∈A∩V by auto
    ultimately have ∃U∈(T{restricted to}A). ∃V∈(T{restricted to}A). C⊆U∧y∈V∧U∩V=0
by auto
  }
    then have ∀x∈⋃(T{restricted to}A)-C. ∃U∈(T{restricted to}A). ∃V∈(T{restricted
to}A). C⊆U∧x∈V∧U∩V=0 by auto
  }
  then have ∀C. C{is closed in}(T{restricted to}A) ⟶ (∀x∈⋃(T{restricted
to}A)-C. ∃U∈(T{restricted to}A). ∃V∈(T{restricted to}A). C⊆U∧x∈V∧U∩V=0)
   by blast
  then show thesis using IsRegular_def by auto
qed

corollary here_regular:
  shows IsRegular {is hereditary} using regular_here IsHer_def by auto

theorem T1_here:
  assumes T{is T_1} A∈Pow(⋃T) shows (T{restricted to}A){is T_1}
proof-
  from assms(2) have un:⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
  {
    fix x y
    assume x∈Ay∈Ax≠y
    with ‹A∈Pow(⋃T)› have x∈⋃Ty∈⋃Tx≠y by auto
    then have ∃U∈T. x∈U∧y∉U using assms(1) isT1_def by auto
    then obtain U where U∈Tx∈Uy∉U by auto
    with ‹x∈A› have A∩U∈(T{restricted to}A) x∈A∩U y∉A∩U unfolding RestrictedTo_def
by auto
    then have ∃U∈(T{restricted to}A). x∈U∧y∉U by blast
  }
  with un have ∀x y. x∈⋃(T{restricted to}A) ∧ y∈⋃(T{restricted to}A)
∧ x≠y ⟶ (∃U∈(T{restricted to}A). x∈U∧y∉U)
    by auto
  then show thesis using isT1_def by auto
qed

corollary here_T1:
  shows isT1 {is hereditary} using T1_here IsHer_def by auto

lemma here_and:
  assumes P {is hereditary} Q {is hereditary}
  shows (λT. P(T) ∧ Q(T)) {is hereditary} using assms unfolding IsHer_def
by auto

corollary here_T3:
  shows isT3 {is hereditary} using here_and[OF here_T1 here_regular]
  unfolding IsHer_def using T3_def_alt by blast
```

```
lemma T2_here:
  assumes T{is T₂} A∈Pow(⋃T) shows (T{restricted to}A){is T₂}
proof-
  from assms(2) have un:⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
  {
    fix x y
    assume x∈Ay∈Ax≠y
    with ‹A∈Pow(⋃T)› have x∈⋃Ty∈⋃Tx≠y by auto
    then have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 using assms(1) isT2_def by
auto
    then obtain U V where U∈T V∈Tx∈Uy∈VU∩V=0 by auto
    with ‹x∈A›‹y∈A› have A∩U∈(T{restricted to}A)A∩V∈(T{restricted to}A)
x∈A∩U y∈A∩V (A∩U)∩(A∩V)=0unfolding RestrictedTo_def by auto
    then have ∃U∈(T{restricted to}A). ∃V∈(T{restricted to}A). x∈U∧y∈V∧U∩V=0
unfolding Bex_def by auto
  }
  with un have ∀x y. x∈⋃(T{restricted to}A) ∧ y∈⋃(T{restricted to}A)
∧ x≠y ⟶ (∃U∈(T{restricted to}A). ∃V∈(T{restricted to}A). x∈U∧y∈V∧U∩V=0)
    by auto
  then show thesis using isT2_def by auto
qed

corollary here_T2:
  shows isT2 {is hereditary} using T2_here IsHer_def by auto

lemma T0_here:
  assumes T{is T₀} A∈Pow(⋃T) shows (T{restricted to}A){is T₀}
proof-
  from assms(2) have un:⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
  {
    fix x y
    assume x∈Ay∈Ax≠y
    with ‹A∈Pow(⋃T)› have x∈⋃Ty∈⋃Tx≠y by auto
    then have ∃U∈T. (x∈U∧y∉U)∨(y∈U∧x∉U) using assms(1) isT0_def by
auto
    then obtain U where U∈T (x∈U∧y∉U)∨(y∈U∧x∉U) by auto
    with ‹x∈A›‹y∈A› have A∩U∈(T{restricted to}A) (x∈A∩U∧y∉A∩U)∨(y∈A∩U∧x∉A∩U)
unfolding RestrictedTo_def by auto
    then have ∃U∈(T{restricted to}A). (x∈U∧y∉U)∨(y∈U∧x∉U) unfolding
Bex_def by auto
  }
  with un have ∀x y. x∈⋃(T{restricted to}A) ∧ y∈⋃(T{restricted to}A)
∧ x≠y ⟶ (∃U∈(T{restricted to}A). (x∈U∧y∉U)∨(y∈U∧x∉U))
    by auto
  then show thesis using isT0_def by auto
qed
```

corollary here_T0:
  shows isT0 {is hereditary} **using** T0_here IsHer_def **by** auto

81.3 Spectrum and anti-properties

The spectrum of a topological property is a class of sets such that all topologies defined over that set have that property.

The spectrum of a property gives us the list of sets for which the property doesn't give any topological information. Being in the spectrum of a topological property is an invariant in the category of sets and function; mening that equipollent sets are in the same spectra.

```
definition Spec (_ {is in the spectrum of} _ 99)
  where Spec(K,P) ≡ ∀T. ((T{is a topology} ∧ ⋃T≈K) ⟶ P(T))

lemma equipollent_spect:
  assumes A≈B B {is in the spectrum of} P
  shows  A {is in the spectrum of} P
proof-
  from assms(2) have ∀T. ((T{is a topology} ∧ ⋃T≈B) ⟶ P(T)) using
Spec_def by auto
  then have ∀T. ((T{is a topology} ∧ ⋃T≈A) ⟶ P(T)) using eqpoll_trans[OF
_ assms(1)] by auto
  then show thesis using Spec_def by auto
qed

theorem eqpoll_iff_spec:
  assumes A≈B
  shows (B {is in the spectrum of} P) ⟷ (A {is in the spectrum of}
P)
proof
  assume B {is in the spectrum of} P
  with assms equipollent_spect show A {is in the spectrum of} P by auto
next
  assume A {is in the spectrum of} P
  moreover
  from assms have B≈A using eqpoll_sym by auto
  ultimately show B {is in the spectrum of} P using equipollent_spect
by auto
qed
```

From the previous statement, we see that the spectrum could be formed only by representative of clases of sets. If AC holds, this means that the spectrum can be taken as a set or class of cardinal numbers.

Here is an example of the spectrum. The proof lies in the indiscrite filter `{A}` that can be build for any set. In this proof, we see that without choice, there is no way to define the sepctrum of a property with cardinals because if a set is not comparable with any ordinal, its cardinal is defined as 0 without the set being empty.

```
theorem T4_spectrum:
  shows (A {is in the spectrum of} isT4) ⟷ A ≲ 1
proof
  assume A {is in the spectrum of} isT4
  then have reg:∀T. ((T{is a topology} ∧ ⋃T≈A) ⟶ (T {is T₄})) us-
ing Spec_def by auto
  {
    assume A≠0
    then obtain x where x∈A by auto
    then have x∈⋃{A} by auto
    moreover
    then have {A} {is a filter on}⋃{A} using IsFilter_def by auto
    moreover
    then have ({A}∪{0}) {is a topology} ∧ ⋃({A}∪{0})=A using top_of_filter
by auto
    then have top:({A}∪{0}) {is a topology} ⋃({A}∪{0})≈A using eqpoll_refl
by auto
    then have ({A}∪{0}) {is T₄} using reg by auto
    then have ({A}∪{0}) {is T₂} using T3_is_T2 topology0.T4_is_T3 topology0_def
top by auto
    ultimately have ⋃{A}={x} using filter_T2_imp_card1[of {A}x] by auto
    then have A={x} by auto
    then have A≈1 using singleton_eqpoll_1 by auto
  }
  moreover
  have A=0 ⟶ A≈0 by auto
  ultimately have A≈1∨A≈0 by blast
  then show A≲1 using empty_lepollI eqpoll_imp_lepoll eq_lepoll_trans
by auto
next
  assume A≲1
  have A=0∨A≠0 by auto
  then obtain E where A=0∨E∈A by auto
  then have A≈0∨E∈A by auto
  with ‹A≲1› have A≈0∨A={E} using lepoll_1_is_sing by auto
  then have A≈0∨A≈1 using singleton_eqpoll_1 by auto
  {
    fix T
    assume AS:T{is a topology}⋃T≈A
    {
      assume A≈0
      with AS have T{is a topology} and empty:⋃T=0 using eqpoll_trans
eqpoll_0_is_0 by auto
      then have T{is T₂} using isT2_def by auto
      then have T{is T₁} using T2_is_T1 by auto
      moreover
      from empty have T⊆{0} by auto
```

```
      with AS(1) have T={0} using empty_open by auto
      from empty have rr:∀A. A{is closed in}T ⟶ A=0 using IsClosed_def
by auto
      have ∃U∈T. ∃V∈T. 0⊆U∧0⊆V∧U∩V=0 using empty_open AS(1) by auto
      with rr have ∀A. A{is closed in}T ⟶ (∀B. B{is closed in}T ∧
A∩B=0 ⟶ (∃U∈T. ∃V∈T. A⊆U∧B⊆V∧U∩V=0))
        by blast
      then have T{is normal} using IsNormal_def by auto
      with ‹T{is T₁}› have T{is T₄} using isT4_def by auto
    }
    moreover
    {
      assume A≈1
      with AS have T{is a topology} and NONempty:⋃T≈1 using eqpoll_trans[of
⋃TA1] by auto
      then have ⋃T≲1 using eqpoll_imp_lepoll by auto
      moreover
      {
        assume ⋃T=0
        then have 0≈⋃T by auto
        with NONempty have 0≈1 using eqpoll_trans by blast
        then have 0=1 using eqpoll_0_is_0 eqpoll_sym by auto
        then have False by auto
      }
      then have ⋃T≠0 by auto
      then obtain R where R∈⋃T by blast
      ultimately have ⋃T={R} using lepoll_1_is_sing by auto
      {
        fix x y
        assume x{is closed in}Ty{is closed in}T x∩y=0
        then have x⊆⋃Ty⊆⋃T using IsClosed_def by auto
        then have x=0∨y=0 using ‹x∩y=0› ‹⋃T={R}› by force
        {
          assume x=0
          then have x⊆0y⊆⋃T using ‹y⊆⋃T› by auto
          moreover
          have 0∈T⋃T∈T using AS(1) IsATopology_def empty_open by auto
          ultimately have ∃U∈T. ∃V∈T. x⊆U∧y⊆V∧U∩V=0 by auto
        }
        moreover
        {
          assume x≠0
          with ‹x=0∨y=0› have y=0 by auto
          then have x⊆⋃Ty⊆0 using ‹x⊆⋃T› by auto
          moreover
          have 0∈T⋃T∈T using AS(1) IsATopology_def empty_open by auto
          ultimately have ∃U∈T. ∃V∈T. x⊆U∧y⊆V∧U∩V=0 by auto
        }
        ultimately
```

```
      have (∃U∈T. ∃V∈T. x ⊆ U ∧ y ⊆ V ∧ U ∩ V = 0) by blast
    }
    then have T{is normal} using IsNormal_def by auto
    moreover
    {
      fix x y
      assume x∈⋃Ty∈⋃Tx≠y
      with ‹⋃T={R}› have False by auto
      then have ∃U∈T. x∈U∧y∉U by auto
    }
    then have T{is T₁} using isT1_def by auto
    ultimately have T{is T₄} using isT4_def by auto
  }
  ultimately have T{is T₄} using ‹A≈0∨A≈1› by auto
}
then have ∀T. (T{is a topology} ∧ ⋃T ≈ A) ⟶ (T{is T₄}) by auto
then show A {is in the spectrum of} isT4 using Spec_def by auto
qed
```

If the topological properties are related, then so are the spectra.

```
lemma P_imp_Q_spec_inv:
  assumes ∀T. T{is a topology} ⟶ (Q(T) ⟶ P(T))  A {is in the spectrum
of} Q
  shows A {is in the spectrum of} P
proof-
  from assms(2) have ∀T. T{is a topology} ∧ ⋃T ≈ A ⟶ Q(T) using Spec_def
by auto
  with assms(1) have ∀T. T{is a topology} ∧ ⋃T ≈ A ⟶ P(T) by auto
  then show thesis using Spec_def by auto
qed
```

Since we already now the spectrum of T_4; if we now the spectrum of T_0, it should be easier to compute the spectrum of T_1, T_2 and T_3.

```
theorem T0_spectrum:
  shows (A {is in the spectrum of} isT0) ⟷ A ≲ 1
proof
  assume A {is in the spectrum of} isT0
  then have reg:∀T. ((T{is a topology} ∧ ⋃T≈A) ⟶ (T {is T₀})) us-
ing Spec_def by auto
  {
    assume A≠0
    then obtain x where x∈A by auto
    then have x∈⋃{A} by auto
    moreover
    then have {A} {is a filter on}⋃{A} using IsFilter_def by auto
    moreover
    then have ({A}∪{0}) {is a topology} ∧ ⋃({A}∪{0})=A using top_of_filter
by auto
```

```
    then have ({A}∪{0}) {is a topology} ∧ ⋃({A}∪{0})≈A using eqpoll_refl
by auto
    then have ({A}∪{0}) {is T0} using reg by auto
    {
      fix y
      assume y∈Ax≠y
      with ‹({A}∪{0}) {is T0}› obtain U where U∈({A}∪{0}) and dis:(x
∈ U ∧ y ∉ U) ∨ (y ∈ U ∧ x ∉ U) using isT0_def by auto
      then have U=A by auto
      with dis ‹y∈A› ‹x∈⋃{A}› have False by auto
    }
    then have ∀y∈A. y=x by auto
    with ‹x∈⋃{A}› have A={x} by blast
    then have A≈1 using singleton_eqpoll_1 by auto
  }
  moreover
  have A=0 ⟶ A≈0 by auto
  ultimately have A≈1∨A≈0 by blast
  then show A≲1 using empty_lepollI eqpoll_imp_lepoll eq_lepoll_trans
by auto
next
  assume A≲1
  {
    fix T
    assume T{is a topology}
    then have (T{is T4})⟶(T{is T0}) using topology0.T4_is_T3 T3_is_T2
T2_is_T1 T1_is_T0
      topology0_def by auto
  }
  then have ∀T. T{is a topology} ⟶ ((T{is T4})⟶(T{is T0})) by auto
  then have (A {is in the spectrum of} isT4) ⟶ (A {is in the spectrum
of} isT0)
    using P_imp_Q_spec_inv[of λT. (T{is T4})λT. T{is T0}] by auto
  then show (A {is in the spectrum of} isT0) using T4_spectrum ‹A≲1›
by auto
qed

theorem T1_spectrum:
  shows (A {is in the spectrum of} isT1) ⟷ A ≲ 1
proof-
  note T2_is_T1 T3_is_T2 topology0.T4_is_T3
  then have (A {is in the spectrum of} isT4) ⟶ (A {is in the spectrum
of} isT1)
    using P_imp_Q_spec_inv[of isT4isT1] topology0_def by auto
  moreover
  note T1_is_T0
  then have (A {is in the spectrum of} isT1) ⟶ (A {is in the spectrum
of}isT0)
    using P_imp_Q_spec_inv[of isT1isT0] by auto
```

```
  moreover
  note T0_spectrum T4_spectrum
  ultimately show thesis by blast
qed

theorem T2_spectrum:
  shows (A {is in the spectrum of} isT2) ⟷ A ≲ 1
proof-
  note T3_is_T2 topology0.T4_is_T3
  then have (A {is in the spectrum of} isT4) ⟶ (A {is in the spectrum
of} isT2)
    using P_imp_Q_spec_inv[of isT4isT2] topology0_def by auto
  moreover
  note T2_is_T1
  then have (A {is in the spectrum of} isT2) ⟶ (A {is in the spectrum
of}isT1)
    using P_imp_Q_spec_inv[of isT2isT1] by auto
  moreover
  note T1_spectrum T4_spectrum
  ultimately show thesis by blast
qed

theorem T3_spectrum:
  shows (A {is in the spectrum of} isT3) ⟷ A ≲ 1
proof-
  note topology0.T4_is_T3
  then have (A {is in the spectrum of} isT4) ⟶ (A {is in the spectrum
of} isT3)
    using P_imp_Q_spec_inv[of isT4isT3] topology0_def by auto
  moreover
  note T3_is_T2
  then have (A {is in the spectrum of} isT3) ⟶ (A {is in the spectrum
of}isT2)
    using P_imp_Q_spec_inv[of isT3isT2] topology0_def by auto
  moreover
  note T2_spectrum T4_spectrum
  ultimately show thesis by blast
qed

theorem compact_spectrum:
  shows (A {is in the spectrum of} (λT. (⋃T) {is compact in}T)) ⟷
Finite(A)
proof
  assume A {is in the spectrum of} (λT. (⋃T) {is compact in}T)
  then have reg:∀T. T{is a topology} ∧ ⋃T≈A ⟶ ((⋃T) {is compact
in}T) using Spec_def by auto
  have Pow(A){is a topology} ∧ ⋃Pow(A)=A using Pow_is_top by auto
  then have Pow(A){is a topology} ∧ ⋃Pow(A)≈A using eqpoll_refl by
auto
```

```
  with reg have A{is compact in}Pow(A) by auto
  moreover
  have {{x}. x∈A}∈Pow(Pow(A)) by auto
  moreover
  have ⋃{{x}. x∈A}=A by auto
  ultimately have ∃N∈FinPow({{x}. x∈A}). A⊆⋃N using IsCompact_def by auto
  then obtain N where N∈FinPow({{x}. x∈A}) A⊆⋃N by auto
  then have N⊆{{x}. x∈A} Finite(N) A⊆⋃N using FinPow_def by auto
  {
    fix t
    assume t∈{{x}. x∈A}
    then obtain x where x∈At={x} by auto
    with ‹A⊆⋃N› have x∈⋃N by auto
    then obtain B where B∈Nx∈B by auto
    with ‹N⊆{{x}. x∈A}› have B={x} by auto
    with ‹t={x}›‹B∈N› have t∈N by auto
  }
  with ‹N⊆{{x}. x∈A}› have N={{x}. x∈A} by auto
  with ‹Finite(N)› have Finite({{x}. x∈A}) by auto
  let B={⟨x,{x}⟩. x∈A}
  have B:A→{{x}. x∈A} unfolding Pi_def function_def by auto
  then have B:bij(A,{{x}. x∈A}) unfolding bij_def inj_def surj_def using apply_equality by auto
  then have A≈{{x}. x∈A} using eqpoll_def by auto
  with ‹Finite({{x}. x∈A})› show Finite(A) using eqpoll_imp_Finite_iff by auto
next
  assume Finite(A)
  {
    fix T assume T{is a topology} ⋃T≈A
    with ‹Finite(A)› have Finite(⋃T) using eqpoll_imp_Finite_iff by auto
    then have Finite(Pow(⋃T)) using Finite_Pow by auto
    moreover
    have T⊆Pow(⋃T) by auto
    ultimately have Finite(T) using subset_Finite by auto
    {
      fix M
      assume M∈Pow(T)⋃T⊆⋃M
      with ‹Finite(T)› have Finite(M) using subset_Finite by auto
      with ‹⋃T⊆⋃M› have ∃N∈FinPow(M). ⋃T⊆⋃N using FinPow_def by auto
    }
    then have (⋃T){is compact in}T unfolding IsCompact_def by auto
  }
  then show A {is in the spectrum of} (λT. (⋃T) {is compact in}T) using Spec_def by auto
qed
```

It is, at least for some people, surprising that the spectrum of some properties cannot be completely determined in *ZF*.

```
theorem compactK_spectrum:
  assumes {the axiom of}K{choice holds for subsets}(Pow(K)) Card(K)
  shows (A {is in the spectrum of} (λT. ((⋃T){is compact of cardinal}
csucc(K){in}T))) ⟷ (A≲K)
proof
  assume A {is in the spectrum of} (λT. ((⋃T){is compact of cardinal}
csucc(K){in}T))
  then have reg:∀T. T{is a topology}∧⋃T≈A ⟶ ((⋃T){is compact of
cardinal} csucc(K){in}T) using Spec_def by auto
  then have A{is compact of cardinal} csucc(K) {in} Pow(A) using Pow_is_top[of
A] by auto
  then have ∀M∈Pow(Pow(A)). A⊆⋃M ⟶ (∃N∈Pow(M). A⊆⋃N ∧ N≺csucc(K))
unfolding IsCompactOfCard_def by auto
  moreover
  have {{x}. x∈A}∈Pow(Pow(A)) by auto
  moreover
  have A=⋃{{x}. x∈A} by auto
  ultimately have ∃N∈Pow({{x}. x∈A}). A⊆⋃N ∧ N≺csucc(K) by auto
  then obtain N where N∈Pow({{x}. x∈A}) A⊆⋃N N≺csucc(K) by auto
  then have N⊆{{x}. x∈A} N≺csucc(K) A⊆⋃N using FinPow_def by auto
  {
    fix t
    assume t∈{{x}. x∈A}
    then obtain x where x∈At={x} by auto
    with ‹A⊆⋃N› have x∈⋃N by auto
    then obtain B where B∈Nx∈B by auto
    with ‹N⊆{{x}. x∈A}› have B={x} by auto
    with ‹t={x}›‹B∈N› have t∈N by auto
  }
  with ‹N⊆{{x}. x∈A}› have N={{x}. x∈A} by auto
  let B={⟨x,{x}⟩. x∈A}
  from ‹N={{x}. x∈A}› have B:A→ N unfolding Pi_def function_def by auto
  with ‹N={{x}. x∈A}› have B:inj(A,N) unfolding inj_def using apply_equality
by auto
  then have A≲N using lepoll_def by auto
  with ‹N≺csucc(K)› have A≺csucc(K) using lesspoll_trans1 by auto
  then show A≲K using Card_less_csucc_eq_le assms(2) by auto
next
  assume A≲K
  {
    fix T
    assume T{is a topology}⋃T≈A
    have Pow(⋃T){is a topology} using Pow_is_top by auto
    {
      fix B
      assume AS:B∈Pow(⋃T)
      then have {{i}. i∈B}⊆{{i} .i∈⋃T} by auto
```

```
      moreover
      have B=⋃{{i}. i∈B} by auto
      ultimately have ∃S∈Pow({{i}. i∈⋃T}). B=⋃S by auto
      then have B∈{⋃U. U∈Pow({{i}. i∈⋃T})} by auto
    }
    moreover
    {
      fix B
      assume AS:B∈{⋃U. U∈Pow({{i}. i∈⋃T})}
      then have B∈Pow(⋃T) by auto
    }
    ultimately
    have base:{{x}. x∈⋃T} {is a base for}Pow(⋃T) unfolding IsAbaseFor_def
by auto
    let f={⟨i,{i}⟩. i∈⋃T}
    have f:f:⋃T→ {{x}. x∈⋃T} using Pi_def function_def by auto
    moreover
    {
      fix w x
      assume as:w∈⋃Tx∈⋃Tfw=fx
      with f have fw={w} fx={x} using apply_equality by auto
      with as(3) have w=x by auto
    }
    with f have f:inj(⋃T,{{x}. x∈⋃T}) unfolding inj_def by auto
    moreover
    {
      fix xa
      assume xa∈{{x}. x∈⋃T}
      then obtain x where x∈⋃Txa={x} by auto
      with f have fx=xa using apply_equality by auto
      with ‹x∈⋃T› have ∃x∈⋃T. fx=xa by auto
    }
    then have ∀xa∈{{x}. x∈⋃T}. ∃x∈⋃T. fx=xa by blast
    ultimately have f:bij(⋃T,{{x}. x∈⋃T}) unfolding bij_def surj_def
by auto
    then have ⋃T≈{{x}. x∈⋃T} using eqpoll_def by auto
    then have {{x}. x∈⋃T}≈⋃T using eqpoll_sym by auto
    with ‹⋃T≈A› have {{x}. x∈⋃T}≈A using eqpoll_trans by blast
    then have {{x}. x∈⋃T}≲A using eqpoll_imp_lepoll by auto
    with ‹A≲K› have {{x}. x∈⋃T}≲K using lepoll_trans by blast
    then have {{x}. x∈⋃T}≺csucc(K) using assms(2) Card_less_csucc_eq_le
by auto
    with base have Pow(⋃T) {is of second type of cardinal}csucc(K) un-
folding IsSecondOfCard_def by auto
    moreover
    have ⋃Pow(⋃T)=⋃T by auto
    with calculation assms(1) ‹Pow(⋃T){is a topology}› have (⋃T) {is
compact of cardinal}csucc(K){in}Pow(⋃T)
      using compact_of_cardinal_Q[of KPow(⋃T)] by auto
```

```
    moreover
    have T⊆Pow(⋃T) by auto
    ultimately have (⋃T) {is compact of cardinal}csucc(K){in}T using
compact_coarser by auto
  }
  then show A {is in the spectrum of} (λT. ((⋃T){is compact of cardinal}csucc(K)
{in}T)) using Spec_def by auto
qed

theorem compactK_spectrum_reverse:
  assumes ∀A. (A {is in the spectrum of} (λT. ((⋃T){is compact of cardinal}
csucc(K){in}T))) ⟷ (A≲K) InfCard(K)
  shows {the axiom of}K{choice holds for subsets}(Pow(K))
proof-
  have K≲K using lepoll_refl by auto
  then have K {is in the spectrum of} (λT. ((⋃T){is compact of cardinal}
csucc(K){in}T)) using assms(1) by auto
  moreover
  have Pow(K){is a topology} using Pow_is_top by auto
  moreover
  have ⋃Pow(K)=K by auto
  then have ⋃Pow(K)≈K using eqpoll_refl by auto
  ultimately
  have K {is compact of cardinal} csucc(K){in}Pow(K) using Spec_def by
auto
  then show thesis using Q_disc_comp_csuccQ_eq_Q_choice_csuccQ assms(2)
by auto
qed
```

This last theorem states that if one of the forms of the axiom of choice related to this compactness property fails, then the spectrum will be different. Notice that even for Lindelöf spaces that will happend.

The spectrum gives us the posibility to define what an anti-property means. A space is anti-`P` if the only subspaces which have the property are the ones in the spectrum of `P`. This concept tries to put together spaces that are completely opposite to spaces where `P(T)`.

```
definition
  antiProperty (_{is anti-}_ 50)
  where T{is anti-}P ≡ ∀A∈Pow(⋃T). P(T{restricted to}A) ⟶ (A {is
in the spectrum of} P)

abbreviation
  ANTI(P) ≡ λT. (T{is anti-}P)
```

A first, very simple, but very useful result is the following: when the properties are related and the spectra are equal, then the anti-properties are related in the oposite direction.

```
theorem (in topology0) eq_spect_rev_imp_anti:
```

```
  assumes ∀T. T{is a topology} ⟶ P(T) ⟶ Q(T) ∀A. (A{is in the spectrum
of}Q) ⟶ (A{is in the spectrum of}P)
    and T{is anti-}Q
  shows T{is anti-}P
proof-
  {
    fix A
    assume A∈Pow(⋃T)P(T{restricted to}A)
    with assms(1) have Q(T{restricted to}A) using Top_1_L4 by auto
    with assms(3) ‹A∈Pow(⋃T)› have A{is in the spectrum of}Q using antiProperty_def
by auto
    with assms(2) have A{is in the spectrum of}P by auto
  }
  then show thesis using antiProperty_def by auto
qed
```

If a space can be `P(T)∧Q(T)` only in case the underlying set is in the spectrum of `P`; then `Q(T)⟶ANTI(P,T)` when `Q` is hereditary.

```
theorem Q_P_imp_Spec:
  assumes ∀T. ((T{is a topology}∧P(T)∧Q(T))⟶ ((⋃T){is in the spectrum
of}P))
    and Q{is hereditary}
  shows ∀T. T{is a topology} ⟶ (Q(T)⟶(T{is anti-}P))
proof
  fix T
  {
    assume T{is a topology}
    {
      assume Q(T)
      {
        assume ¬(T{is anti-}P)
        then obtain A where A∈Pow(⋃T) P(T{restricted to}A)¬(A{is in
the spectrum of}P)
          unfolding antiProperty_def by auto
        from ‹Q(T)›‹T{is a topology}›‹A∈Pow(⋃T)› assms(2) have Q(T{restricted
to}A)
          unfolding IsHer_def by auto
        moreover
        note ‹P(T{restricted to}A)› assms(1)
        moreover
        from ‹T{is a topology}› have (T{restricted to}A){is a topology}
using topology0.Top_1_L4
          topology0_def by auto
        moreover
        from ‹A∈Pow(⋃T)› have ⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
        ultimately have A{is in the spectrum of}P by auto
        with ‹¬(A{is in the spectrum of}P)› have False by auto
      }
```

```
      then have T{is anti-}P by auto
    }
    then have Q(T)⟶(T{is anti-}P) by auto
  }
  then show (T {is a topology}) ⟶ (Q(T) ⟶ (T{is anti-}P)) by auto
qed
```

If a topologycal space has an hereditary property, then it has its double-anti property.

```
theorem (in topology0)her_P_imp_anti2P:
  assumes P{is hereditary} P(T)
  shows T{is anti-}ANTI(P)
proof-
  {
    assume ¬(T{is anti-}ANTI(P))
    then have ∃A∈Pow(⋃T). ((T{restricted to}A){is anti-}P)∧¬(A{is in
the spectrum of}ANTI(P))
      unfolding antiProperty_def[of _ ANTI(P)] by auto
    then obtain A where A_def:A∈Pow(⋃T)¬(A{is in the spectrum of}ANTI(P))(T{restricted
to}A){is anti-}P
      by auto
    from ‹A∈Pow(⋃T)› have tot:⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
    from A_def have reg:∀B∈Pow(⋃(T{restricted to}A)). P((T{restricted
to}A){restricted to}B) ⟶ (B{is in the spectrum of}P)
      unfolding antiProperty_def by auto
    have ∀B∈Pow(A). (T{restricted to}A){restricted to}B=T{restricted
to}B using subspace_of_subspace ‹A∈Pow(⋃T)› by auto
    then have ∀B∈Pow(A). P(T{restricted to}B) ⟶ (B{is in the spectrum
of}P) using reg tot
      by force
    moreover
    have ∀B∈Pow(A). P(T{restricted to}B) using assms ‹A∈Pow(⋃T)› un-
folding IsHer_def using topSpaceAssum by blast
    ultimately have reg2:∀B∈Pow(A). (B{is in the spectrum of}P) by auto
    from ‹¬(A{is in the spectrum of}ANTI(P))› have ∃T. T{is a topology}
∧ ⋃T≈A ∧ ¬(T{is anti-}P)
      unfolding Spec_def by auto
    then obtain S where S{is a topology} ⋃S≈A ¬(S{is anti-}P) by auto
    from ‹¬(S{is anti-}P)› have ∃B∈Pow(⋃S). P(S{restricted to}B) ∧
¬(B{is in the spectrum of}P) unfolding antiProperty_def by auto
    then obtain B where B_def:¬(B{is in the spectrum of}P) B∈Pow(⋃S)
by auto
    then have B≲⋃S using subset_imp_lepoll by auto
    with ‹⋃S≈A› have B≲A using lepoll_eq_trans by auto
    then obtain f where f∈inj(B,A) unfolding lepoll_def by auto
    then have f∈bij(B,range(f)) using inj_bij_range by auto
    then have B≈range(f) unfolding eqpoll_def by auto
    with B_def(1) have ¬(range(f){is in the spectrum of}P) using eqpoll_iff_spec
```

```
by auto
    moreover
    with ‹f∈inj(B,A)› have range(f)⊆A unfolding inj_def Pi_def by auto
    with reg2 have range(f){is in the spectrum of}P by auto
    ultimately have False by auto
  }
  then show thesis by auto
qed
```

The anti-properties are always hereditary

```
theorem anti_here:
  shows ANTI(P){is hereditary}
proof-
  {
    fix T
    assume T {is a topology}ANTI(P,T)
    {
      fix A
      assume A∈Pow(⋃T)
      then have ⋃(T{restricted to}A)=A unfolding RestrictedTo_def by
auto
      moreover
      {
        fix B
        assume B∈Pow(A)P((T{restricted to}A){restricted to}B)
        with ‹A∈Pow(⋃T)› have B∈Pow(⋃T)P(T{restricted to}B) using subspace_of_subspace
by auto
        with ‹ANTI(P,T)› have B{is in the spectrum of}P unfolding antiProperty_def
by auto
      }
      ultimately have ∀B∈Pow(⋃(T{restricted to}A)). (P((T{restricted
to}A){restricted to}B)) ⟶ (B{is in the spectrum of}P)
        by auto
      then have ANTI(P,(T{restricted to}A)) unfolding antiProperty_def
by auto
    }
    then have ∀A∈Pow(⋃T). ANTI(P,(T{restricted to}A)) by auto
  }
  then show thesis using IsHer_def by auto
qed

corollary (in topology0) anti_imp_anti3:
  assumes T{is anti-}P
  shows T{is anti-}ANTI(ANTI(P))
  using anti_here her_P_imp_anti2P assms by auto
```

In the article [5], we can find some results on anti-properties.

```
theorem (in topology0) anti_T0:
  shows (T{is anti-}isT0) ⟷ T={0,⋃T}
```

```
proof
  assume T={0,⋃T}
  {
    fix A
    assume A∈Pow(⋃T)(T{restricted to}A) {is T_0}
    {
      fix B
      assume B∈T{restricted to}A
      then obtain S where S∈T and B=A∩S unfolding RestrictedTo_def by auto
      with ‹T={0,⋃T}› have S∈{0,⋃T} by auto
      then have S=0∨S=⋃T by auto
      with ‹B=A∩S›‹A∈Pow(⋃T)› have B=0∨B=A by auto
    }
    moreover
    {
      have 0∈{0,⋃T} ⋃T∈{0,⋃T} by auto
      with ‹T={0,⋃T}› have 0∈T(⋃T)∈T by auto
      then have A∩0∈(T{restricted to}A) A∩(⋃T)∈(T{restricted to}A) using RestrictedTo_def by auto
      moreover
      from ‹A∈Pow(⋃T)› have A∩(⋃T)=A by auto
      ultimately have 0∈(T{restricted to}A) A∈(T{restricted to}A) by auto
    }
    ultimately have (T{restricted to}A)={0,A} by auto
    with ‹(T{restricted to}A) {is T_0}› have {0,A} {is T_0} by auto
    {
      assume A≠0
      then obtain x where x∈A by blast
      {
        fix y
        assume y∈Ax≠y
        with ‹{0,A} {is T_0}› obtain U where U∈{0,A} and dis:(x ∈ U ∧ y ∉ U) ∨ (y ∈ U ∧ x ∉ U) using isT0_def by auto
        then have U=A by auto
        with dis ‹y∈A› ‹x∈A› have False by auto
      }
      then have ∀y∈A. y=x by auto
      with ‹x∈A› have A={x} by blast
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
      then have A{is in the spectrum of}isT0 using T0_spectrum by auto

    }
    moreover
    {
      assume A=0
      then have A≈0 by auto
```

```
      then have A≲1 using empty_lepollI eq_lepoll_trans by auto
      then have A{is in the spectrum of}isT0 using T0_spectrum by auto
    }
    ultimately have A{is in the spectrum of}isT0 by auto
  }
  then show T{is anti-}isT0 using antiProperty_def by auto
next
  assume T{is anti-}isT0
  then have ∀A∈Pow(⋃T). (T{restricted to}A){is T_0} ⟶ (A{is in the spectrum of}isT0) using antiProperty_def by auto
  then have reg:∀A∈Pow(⋃T). (T{restricted to}A){is T_0} ⟶ (A≲1) using T0_spectrum by auto
  {
    assume ∃A∈T. A≠0∧ A≠⋃T
    then obtain A where A∈TA≠0A≠⋃T by auto
    then obtain x y where x∈A y∈⋃T-A by blast
    with ‹A∈T› have s:{x,y}∈Pow(⋃T) x≠y by auto
    note s
    moreover
    {
      fix b1 b2
      assume b1∈⋃(T{restricted to}{x,y})b2∈⋃(T{restricted to}{x,y})b1≠b2
      moreover
      from s have ⋃(T{restricted to}{x,y})={x,y} unfolding RestrictedTo_def by auto
      ultimately have (b1=x∧b2=y)∨(b1=y∧b2=x) by auto
      with ‹x≠y› have (b1∈{x}∧b2∉{x}) ∨ (b2∈{x}∧b1∉{x}) by auto
      moreover
      from ‹y∈⋃T-A›‹x∈A› have {x}={x,y}∩A by auto
      with ‹A∈T› have {x}∈(T{restricted to}{x,y}) unfolding RestrictedTo_def by auto
      ultimately have ∃U∈(T{restricted to}{x,y}). (b1∈U∧b2∉U) ∨ (b2∈U∧b1∉U) by auto
    }
    then have (T{restricted to}{x,y}){is T_0} using isT0_def by auto
    ultimately have {x,y}≲1 using reg by auto
    moreover
    have x∈{x,y} by auto
    ultimately have {x,y}={x} using lepoll_1_is_sing[of {x,y}x] by auto
    moreover
    have y∈{x,y} by auto
    ultimately have y∈{x} by auto
    then have y=x by auto
    with ‹x≠y› have False by auto
  }
  then have T⊆{0,⋃T} by auto
  moreover
  from topSpaceAssum have 0∈T⋃T∈T using IsATopology_def empty_open by auto
```

```
  ultimately show T={0,⋃T} by auto
qed

lemma indiscrete_spectrum:
  shows (A {is in the spectrum of}(λT. T={0,⋃T})) ⟷ A≲1
proof
  assume (A {is in the spectrum of}(λT. T={0,⋃T}))
  then have reg:∀T. ((T{is a topology} ∧ ⋃T≈A) ⟶ T ={0,⋃T}) using
Spec_def by auto
  moreover
  have ⋃Pow(A)=A by auto
  then have ⋃Pow(A)≈A by auto
  moreover
  have Pow(A) {is a topology} using Pow_is_top by auto
  ultimately have P:Pow(A)={0,A} by auto
  {
    assume A≠0
    then obtain x where x∈A by blast
    then have {x}∈Pow(A) by auto
    with P have {x}=A by auto
    then have A≈1 using singleton_eqpoll_1 by auto
    then have A≲1 using eqpoll_imp_lepoll by auto
  }
  moreover
  {
    assume A=0
    then have A≈0 by auto
    then have A≲1 using empty_lepollI eq_lepoll_trans by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T{is a topology}⋃T≈A
    {
      assume A=0
      with ‹⋃T≈A› have ⋃T≈0 by auto
      then have ⋃T=0 using eqpoll_0_is_0 by auto
      then have T⊆{0} by auto
      with ‹T{is a topology}› have T={0} using empty_open by auto
      then have T={0,⋃T} by auto
    }
    moreover
    {
      assume A≠0
      then obtain E where E∈A by blast
      with ‹A≲1› have A={E} using lepoll_1_is_sing by auto
      then have A≈1 using singleton_eqpoll_1 by auto
```

```
      with ‹⋃T≈A› have NONempty:⋃T≈1 using eqpoll_trans by blast
      then have ⋃T≲1 using eqpoll_imp_lepoll by auto
      moreover
      {
        assume ⋃T=0
        then have 0≈⋃T by auto
        with NONempty have 0≈1 using eqpoll_trans by blast
        then have 0=1 using eqpoll_0_is_0 eqpoll_sym by auto
        then have False by auto
      }
      then have ⋃T≠0 by auto
      then obtain R where R∈⋃T by blast
      ultimately have ⋃T={R} using lepoll_1_is_sing by auto
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow({R}) by auto
      then have T⊆{0,{R}} by blast
      moreover
      with ‹T{is a topology}› have 0∈T⋃T∈T using IsATopology_def by
auto
      moreover
      note ‹⋃T={R}›
      ultimately have T={0,⋃T} by auto
    }
    ultimately have T={0,⋃T} by auto
  }
  then show A {is in the spectrum of}(λT. T={0,⋃T}) using Spec_def by
auto
qed

theorem (in topology0) anti_indiscrete:
  shows (T{is anti-}(λT. T={0,⋃T})) ⟷ T{is T₀}
proof
  assume T{is T₀}
  {
    fix A
    assume A∈Pow(⋃T)T{restricted to}A={0,⋃(T{restricted to}A)}
    then have un:⋃(T{restricted to}A)=A T{restricted to}A={0,A} using
RestrictedTo_def by auto
    from ‹T{is T₀}›‹A∈Pow(⋃T)› have (T{restricted to}A){is T₀} using
T0_here by auto
    {
      assume A=0
      then have A≈0 by auto
      then have A≲1 using empty_lepollI eq_lepoll_trans by auto
    }
    moreover
    {
      assume A≠0
```

```
      then obtain E where E∈A by blast
      {
        fix y
        assume y∈Ay≠E
        with ‹E∈A› un have y∈⋃(T{restricted to}A)E∈⋃(T{restricted to}A)
by auto
        with ‹(T{restricted to}A){is T_0}›‹y≠E› have ∃U∈(T{restricted
to}A). (E∈U∧y∉U)∨(E∉U∧y∈U)
          unfolding isT0_def by blast
        then obtain U where U∈(T{restricted to}A) (E∈U∧y∉U)∨(E∉U∧y∈U)
by auto
        with ‹T{restricted to}A={0,A}› have U=0∨U=A by auto
        with ‹(E∈U∧y∉U)∨(E∉U∧y∈U)›‹y∈A›‹E∈A› have False by auto
      }
      then have ∀y∈A. y=E by auto
      with ‹E∈A› have A={E} by blast
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
    }
    ultimately have A≲1 by auto
    then have A{is in the spectrum of}(λT. T={0,⋃T}) using indiscrete_spectrum
by auto
  }
  then show T{is anti-}(λT. T={0,⋃T}) unfolding antiProperty_def by
auto
next
  assume T{is anti-}(λT. T={0,⋃T})
  then have ∀A∈Pow(⋃T). (T{restricted to}A)={0,⋃(T{restricted to}A)}
⟶ (A {is in the spectrum of} (λT. T={0,⋃T})) using antiProperty_def
by auto
  then have ∀A∈Pow(⋃T). (T{restricted to}A)={0,⋃(T{restricted to}A)}
⟶ A≲1 using indiscrete_spectrum by auto
  moreover
  have ∀A∈Pow(⋃T). ⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
  ultimately have reg:∀A∈Pow(⋃T). (T{restricted to}A)={0,A} ⟶ A≲1
by auto
  {
    fix x y
    assume x∈⋃Ty∈⋃Tx≠y
    {
      assume ∀U∈T. (x∈U∧y∈U)∨(x∉U∧y∉U)
      then have T{restricted to}{x,y}⊆{0,{x,y}} unfolding RestrictedTo_def
by auto
      moreover
      from ‹x∈⋃T›‹y∈⋃T› have emp:0∈T{x,y}∩0=0 and tot: {x,y}={x,y}∩⋃T
⋃T∈T using topSpaceAssum empty_open IsATopology_def by auto
      from emp have 0∈T{restricted to}{x,y} unfolding RestrictedTo_def
by auto
```

```
      moreover
      from tot have {x,y}∈T{restricted to}{x,y} unfolding RestrictedTo_def
by auto
      ultimately have T{restricted to}{x,y}={0,{x,y}} by auto
      with reg ‹x∈⋃T›‹y∈⋃T› have {x,y}≲1 by auto
      moreover
      have x∈{x,y} by auto
      ultimately have {x,y}={x} using lepoll_1_is_sing[of {x,y}x] by auto
      moreover
      have y∈{x,y} by auto
      ultimately have y∈{x} by auto
      then have y=x by auto
      then have False using ‹x≠y› by auto
    }
    then have ∃U∈T. (x∉U∨y∉U)∧(x∈U∨y∈U) by auto
    then have ∃U∈T. (x∈U∧y∉U)∨(x∉U∧y∈U) by auto
  }
  then have ∀x y. x∈⋃T∧y∈⋃T∧ x≠y ⟶ (∃U∈T. (x∈U∧y∉U)∨(y∈U∧x∉U))
by auto
  then show T {is T₀} using isT0_def by auto
qed
```

The conclusion is that being T_0 is just the opposite to being indiscrete.

Next, let's compute the anti-T_i for $i = 1,\ 2,\ 3$ or 4. Surprisingly, they are all the same. Meaning, that the total negation of T_1 is enough to negate all of these axioms.

```
theorem anti_T1:
  shows (T{is anti-}isT1) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
proof
  assume T{is anti-}isT1
  let r={⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}
  have antisym(r) unfolding antisym_def by auto
  moreover
  have trans(r) unfolding trans_def by auto
  moreover
  {
    fix A B
    assume A∈TB∈T
    {
      assume ¬(A⊆B∨B⊆A)
      then have A-B≠0B-A≠0 by auto
      then obtain x y where x∈Ax∉By∈By∉A x≠y by blast
      then have {x,y}∩A={x}{x,y}∩B={y} by auto
      moreover
      from ‹A∈T›‹B∈T› have {x,y}∩A∈T{restricted to}{x,y}{x,y}∩B∈T{restricted
to}{x,y} unfolding
        RestrictedTo_def by auto
```

```
      ultimately have open_set:{x}∈T{restricted to}{x,y}{y}∈T{restricted
to}{x,y} by auto
      have x∈⋃Ty∈⋃T using ‹A∈T›‹B∈T›‹x∈A›‹y∈B› by auto
      then have sub:{x,y}∈Pow(⋃T) by auto
      then have tot:⋃(T{restricted to}{x,y})={x,y} unfolding RestrictedTo_def
by auto
      {
        fix s t
        assume s∈⋃(T{restricted to}{x,y})t∈⋃(T{restricted to}{x,y})s≠t
        with tot have s∈{x,y}t∈{x,y}s≠t by auto
        then have (s=x∧t=y)∨(s=y∧t=x) by auto
        with open_set have ∃U∈(T{restricted to}{x,y}). s∈U∧t∉U using
‹x≠y› by auto
      }
      then have (T{restricted to}{x,y}){is T₁} unfolding isT1_def by
auto
      with sub ‹T{is anti-}isT1› tot have {x,y} {is in the spectrum of}isT1
using antiProperty_def
        by auto
      then have {x,y}≲1 using T1_spectrum by auto
      moreover
      have x∈{x,y} by auto
      ultimately have {x}={x,y} using lepoll_1_is_sing[of {x,y}x] by auto
      moreover
      have y∈{x,y} by auto
      ultimately
      have y∈{x} by auto
      then have x=y by auto
      then have False using ‹x∈A›‹y∉A› by auto
    }
    then have A⊆B∨B⊆A by auto
  }
  then have r {is total on}T using IsTotal_def by auto
  ultimately
  show IsLinOrder(T,r) using IsLinOrder_def by auto
next
  assume IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})
  then have ordTot:∀S∈T. ∀B∈T. S⊆B∨B⊆S unfolding IsLinOrder_def IsTotal_def
by auto
  {
    fix A
    assume A∈Pow(⋃T) and T1:(T{restricted to}A) {is T₁}
    then have tot:⋃(T{restricted to}A)=A unfolding RestrictedTo_def by
auto
    {
      fix U V
      assume U∈T{restricted to}AV∈T{restricted to}A
      then obtain AU AV where AU∈TAV∈TU=A∩AUV=A∩AV unfolding RestrictedTo_def
by auto
```

```
      with ordTot have U⊆V∨V⊆U by auto
    }
    then have ordTotSub:∀S∈T{restricted to}A. ∀B∈T{restricted to}A.
S⊆B∨B⊆S by auto
    {
      assume A=0
      then have A≈0 by auto
      moreover
      have 0≲1 using empty_lepollI by auto
      ultimately have A≲1 using eq_lepoll_trans by auto
      then have A{is in the spectrum of}isT1 using T1_spectrum by auto
    }
    moreover
    {
      assume A≠0
      then obtain t where t∈A by blast
      {
        fix y
        assume y∈Ay≠t
        with ‹t∈A› tot T1 obtain U where U∈(T{restricted to}A)y∈Ut∉U
unfolding isT1_def
          by auto
        from ‹y≠t› have t≠y by auto
        with ‹y∈A›‹t∈A› tot T1 obtain V where V∈(T{restricted to}A)t∈Vy∉V
unfolding isT1_def
          by auto
        with ‹y∈U›‹t∉U› have ¬(U⊆V∨V⊆U) by auto
        with ordTotSub ‹U∈(T{restricted to}A)›‹V∈(T{restricted to}A)›
have False by auto
      }
      then have ∀y∈A. y=t by auto
      with ‹t∈A› have A={t} by blast
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
      then have A{is in the spectrum of}isT1 using T1_spectrum by auto
    }
    ultimately
    have A{is in the spectrum of}isT1 by auto
  }
  then show T{is anti-}isT1 using antiProperty_def by auto
qed

corollary linordtop_here:
  shows (λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})){is hereditary}
  using anti_T1 anti_here[of isT1] by auto

theorem (in topology0) anti_T4:
  shows (T{is anti-}isT4) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
```

```
proof
  assume T{is anti-}isT4
  let r={⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}
  have antisym(r) unfolding antisym_def by auto
  moreover
  have trans(r) unfolding trans_def by auto
  moreover
  {
    fix A B
    assume A∈TB∈T
    {
      assume ¬(A⊆B∨B⊆A)
      then have A-B≠0B-A≠0 by auto
      then obtain x y where x∈Ax∉By∈By∉A x≠y by blast
      then have {x,y}∩A={x}{x,y}∩B={y} by auto
      moreover
      from ‹A∈T›‹B∈T› have {x,y}∩A∈T{restricted to}{x,y}{x,y}∩B∈T{restricted
to}{x,y} unfolding
        RestrictedTo_def by auto
      ultimately have open_set:{x}∈T{restricted to}{x,y}{y}∈T{restricted
to}{x,y} by auto
      have x∈⋃Ty∈⋃T using ‹A∈T›‹B∈T›‹x∈A›‹y∈B› by auto
      then have sub:{x,y}∈Pow(⋃T) by auto
      then have tot:⋃(T{restricted to}{x,y})={x,y} unfolding RestrictedTo_def
by auto
      {
        fix s t
        assume s∈⋃(T{restricted to}{x,y})t∈⋃(T{restricted to}{x,y})s≠t
        with tot have s∈{x,y}t∈{x,y}s≠t by auto
        then have (s=x∧t=y)∨(s=y∧t=x) by auto
        with open_set have ∃U∈(T{restricted to}{x,y}). s∈U∧t∉U using
‹x≠y› by auto
      }
      then have (T{restricted to}{x,y}){is T₁} unfolding isT1_def by
auto
      moreover
      {
        fix s
        assume AS:s{is closed in}(T{restricted to}{x,y})
        {
          fix t
          assume AS2:t{is closed in}(T{restricted to}{x,y})s∩t=0
          have (T{restricted to}{x,y}){is a topology} using Top_1_L4 by
auto
          with tot have 0∈(T{restricted to}{x,y}){x,y}∈(T{restricted
to}{x,y}) using empty_open
            union_open[where 𝒜=T{restricted to}{x,y}] by auto
          moreover
          note open_set
```

```
          moreover
          have T{restricted to}{x,y}⊆Pow(⋃(T{restricted to}{x,y})) by
blast
          with tot have T{restricted to}{x,y}⊆Pow({x,y}) by auto
          ultimately have T{restricted to}{x,y}={0,{x},{y},{x,y}} by blast
          moreover have {0,{x},{y},{x,y}}=Pow({x,y}) by blast
          ultimately have P:T{restricted to}{x,y}=Pow({x,y}) by simp
          with tot have {A∈Pow({x,y}). A{is closed in}(T{restricted to}{x,y})}={A
∈ Pow({x, y}) . A ⊆ {x, y} ∧ {x, y} - A ∈ Pow({x, y})} using IsClosed_def
by simp
          with P have S:{A∈Pow({x,y}). A{is closed in}(T{restricted to}{x,y})}=T{restricted
to}{x,y} by auto
          from AS AS2(1) have s∈Pow({x,y}) t∈Pow({x,y}) using IsClosed_def
tot by auto
          moreover
          note AS2(1) AS
          ultimately have s∈{A∈Pow({x,y}). A{is closed in}(T{restricted
to}{x,y})}t∈{A∈Pow({x,y}). A{is closed in}(T{restricted to}{x,y})}
            by auto
          with S AS2(2) have s∈T{restricted to}{x,y} t∈T{restricted to}{x,y}s∩t=0
by auto
          then have ∃U∈(T{restricted to}{x,y}). ∃V∈(T{restricted to}{x,y}).
s⊆U∧t⊆V∧U∩V=0 by auto
        }
        then have ∀t. t{is closed in}(T{restricted to}{x,y})∧s∩t=0 ⟶
(∃U∈(T{restricted to}{x,y}). ∃V∈(T{restricted to}{x,y}). s⊆U∧t⊆V∧U∩V=0)
          by auto
      }
      then have ∀s. s{is closed in}(T{restricted to}{x,y}) ⟶ (∀t. t{is
closed in}(T{restricted to}{x,y})∧s∩t=0 ⟶ (∃U∈(T{restricted to}{x,y}).
∃V∈(T{restricted to}{x,y}). s⊆U∧t⊆V∧U∩V=0))
        by auto
      then have (T{restricted to}{x,y}){is normal} using IsNormal_def
by auto
      ultimately have (T{restricted to}{x,y}){is T₄} using isT4_def by
auto
      with sub ‹T{is anti-}isT4› tot have {x,y} {is in the spectrum of}isT4
using antiProperty_def
        by auto
      then have {x,y}≲1 using T4_spectrum by auto
      moreover
      have x∈{x,y} by auto
      ultimately have {x}={x,y} using lepoll_1_is_sing[of {x,y}x] by auto
      moreover
      have y∈{x,y} by auto
      ultimately
      have y∈{x} by auto
      then have x=y by auto
      then have False using ‹x∈A›‹y∉A› by auto
```

```
    }
    then have A⊆B∨B⊆A by auto
  }
  then have r {is total on}T using IsTotal_def by auto
  ultimately
  show IsLinOrder(T,r) using IsLinOrder_def by auto
next
  assume IsLinOrder(T, {⟨U,V⟩ ∈ Pow(⋃T) × Pow(⋃T) . U ⊆ V})
  then have T{is anti-}isT1 using anti_T1 by auto
  moreover
  have ∀T. T{is a topology} ⟶ (T{is T₄}) ⟶ (T{is T₁}) using topology0.T4_is_T3

    T3_is_T2 T2_is_T1 topology0_def by auto
  moreover
  have  ∀A. (A {is in the spectrum of} isT1) ⟶ (A {is in the spectrum
of} isT4) using T1_spectrum T4_spectrum
    by auto
  ultimately show T{is anti-}isT4 using eq_spect_rev_imp_anti[of isT4isT1]
by auto
qed

theorem (in topology0) anti_T3:
  shows (T{is anti-}isT3) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
proof
  assume T{is anti-}isT3
  moreover
  have ∀T. T{is a topology} ⟶ (T{is T₄}) ⟶ (T{is T₃}) using topology0.T4_is_T3

    topology0_def by auto
  moreover
  have  ∀A. (A {is in the spectrum of} isT3) ⟶ (A {is in the spectrum
of} isT4) using T3_spectrum T4_spectrum
    by auto
  ultimately have T{is anti-}isT4 using eq_spect_rev_imp_anti[of isT4isT3]
by auto
  then show IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}) using anti_T4
by auto
next
  assume IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})
  then have T{is anti-}isT1 using anti_T1 by auto
  moreover
  have ∀T. T{is a topology} ⟶ (T{is T₃}) ⟶ (T{is T₁}) using
    T3_is_T2 T2_is_T1 topology0_def by auto
  moreover
  have  ∀A. (A {is in the spectrum of} isT1) ⟶ (A {is in the spectrum
of} isT3) using T1_spectrum T3_spectrum
    by auto
  ultimately show T{is anti-}isT3 using eq_spect_rev_imp_anti[of isT3isT1]
```

```
by auto
qed

theorem (in topology0) anti_T2:
  shows (T{is anti-}isT2) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
proof
  assume T{is anti-}isT2
  moreover
  have ∀T. T{is a topology} ⟶ (T{is T₄}) ⟶ (T{is T₂}) using topology0.T4_is_T3

    T3_is_T2 topology0_def by auto
  moreover
  have  ∀A. (A {is in the spectrum of} isT2) ⟶ (A {is in the spectrum
of} isT4) using T2_spectrum T4_spectrum
    by auto
  ultimately have T{is anti-}isT4 using eq_spect_rev_imp_anti[of isT4isT2]
by auto
  then show IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}) using anti_T4
by auto
next
  assume IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})
  then have T{is anti-}isT1 using anti_T1 by auto
  moreover
  have ∀T. T{is a topology} ⟶ (T{is T₂}) ⟶ (T{is T₁}) using T2_is_T1
by auto
  moreover
  have  ∀A. (A {is in the spectrum of} isT1) ⟶ (A {is in the spectrum
of} isT2) using T1_spectrum T2_spectrum
    by auto
  ultimately show T{is anti-}isT2 using eq_spect_rev_imp_anti[of isT2isT1]
by auto
qed

lemma linord_spectrum:
  shows (A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))) ⟷ A≲1
proof
  assume A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
  then have reg:∀T. T{is a topology}∧ ⋃T≈A ⟶ IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V})
    using Spec_def by auto
  {
    assume A=0
    moreover
    have 0≲1 using empty_lepollI by auto
    ultimately have A≲1 using eq_lepoll_trans by auto
  }
```

```
  moreover
  {
    assume A≠0
    then obtain x where x∈A by blast
    moreover
    {
      fix y
      assume y∈A
      have Pow(A) {is a topology} using Pow_is_top by auto
      moreover
      have ⋃Pow(A)=A by auto
      then have ⋃Pow(A)≈A by auto
      note reg
      ultimately have IsLinOrder(Pow(A),{⟨U,V⟩∈Pow(⋃Pow(A))×Pow(⋃Pow(A)).
U⊆V}) by auto
      then have IsLinOrder(Pow(A),{⟨U,V⟩∈Pow(A)×Pow(A). U⊆V}) by auto
      with ‹x∈A›‹y∈A› have {x}⊆{y}∨{y}⊆{x} unfolding IsLinOrder_def
IsTotal_def by auto
      then have x=y by auto
    }
    ultimately have A={x} by blast
    then have A≈1 using singleton_eqpoll_1 by auto
    then have A≲1 using eqpoll_imp_lepoll by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  then have ind:A{is in the spectrum of}(λT. T={0,⋃T}) using indiscrete_spectrum
by auto
  {
    fix T
    assume AS:T{is a topology} T={0,⋃T}
    have trans({⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}) unfolding trans_def by
auto
    moreover
    have antisym({⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}) unfolding antisym_def
by auto
    moreover
    have {⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}{is total on}T
    proof-
      {
        fix aa b
        assume aa∈Tb∈T
        with AS(2) have aa∈{0,⋃T}b∈{0,⋃T} by auto
        then have aa=0∨aa=⋃Tb=0∨b=⋃T by auto
        then have aa⊆b∨b⊆aa by auto
        then have ⟨aa, b⟩ ∈ Collect(Pow(⋃T) × Pow(⋃T), split((⊆)))
∨ ⟨b, aa⟩ ∈ Collect(Pow(⋃T) × Pow(⋃T), split((⊆)))
        using ‹aa∈T›‹b∈T› by auto
```

```
      }
      then show thesis using IsTotal_def by auto
    qed
    ultimately have IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}) un-
folding IsLinOrder_def by auto
  }
  then have  ∀T. T {is a topology} ⟶ T = {0, ⋃T} ⟶ IsLinOrder(T,
{⟨U,V⟩ ∈ Pow(⋃T) × Pow(⋃T) . U ⊆ V}) by auto
  then show A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
    using P_imp_Q_spec_inv[of λT. T={0,⋃T}λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V})]
    ind by auto
qed

theorem (in topology0) anti_linord:
  shows (T{is anti-}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})))
⟷ T{is T₁}
proof
  assume AS:T{is anti-}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V}))
  {
    assume ¬(T{is T₁})
    then obtain x y where x∈⋃Ty∈⋃Tx≠y∀U∈T. x∉U∨y∈U unfolding isT1_def
by auto
    {
      assume {x}∈T{restricted to}{x,y}
      then obtain U where U∈T {x}={x,y}∩U unfolding RestrictedTo_def
by auto
      moreover
      have x∈{x} by auto
      ultimately have U∈Tx∈U by auto
      moreover
      {
        assume y∈U
        then have y∈{x,y}∩U by auto
        with ‹{x}={x,y}∩U› have y∈{x} by auto
        with ‹x≠y› have False by auto
      }
      then have y∉U by auto
      moreover
      note ‹∀U∈T. x∉U∨y∈U›
      ultimately have False by auto
    }
    then have {x}∉T{restricted to}{x,y} by auto
    moreover
    have tot:⋃(T{restricted to}{x,y})={x,y} using ‹x∈⋃T›‹y∈⋃T› un-
folding RestrictedTo_def by auto
    moreover
    have T{restricted to}{x,y}⊆Pow(⋃(T{restricted to}{x,y})) by auto
```

```
    ultimately have T{restricted to}{x,y}⊆Pow({x,y})-{{x}} by auto
    moreover
    have Pow({x,y})={0,{x,y},{x},{y}} by blast
    ultimately have T{restricted to}{x,y}⊆{0,{x,y},{y}} by auto
    moreover
    have IsLinOrder({0,{x,y},{y}},{⟨U,V⟩∈Pow({x,y})×Pow({x,y}). U⊆V})
    proof-
      have antisym(Collect(Pow({x, y}) × Pow({x, y}), split((⊆)))) us-
ing antisym_def by auto
      moreover
      have trans(Collect(Pow({x, y}) × Pow({x, y}), split((⊆)))) us-
ing trans_def by auto
      moreover
      have Collect(Pow({x, y}) × Pow({x, y}), split((⊆))) {is total on}
{0, {x, y}, {y}} using IsTotal_def by auto
      ultimately show IsLinOrder({0,{x,y},{y}},{⟨U,V⟩∈Pow({x,y})×Pow({x,y}).
U⊆V}) using IsLinOrder_def by auto
    qed
    ultimately have IsLinOrder(T{restricted to}{x,y},{⟨U,V⟩∈Pow({x,y})×Pow({x,y}).
U⊆V}) using ord_linear_subset
      by auto
    with tot have IsLinOrder(T{restricted to}{x,y},{⟨U,V⟩∈Pow(⋃(T{restricted
to}{x,y}))×Pow(⋃(T{restricted to}{x,y})). U⊆V})
      by auto
    then have IsLinOrder(T{restricted to}{x,y},Collect(Pow(⋃(T {restricted
to} {x,y})) × Pow(⋃(T {restricted to} {x,y})), split((⊆)))) by auto
    moreover
    from ‹x∈⋃T›‹y∈⋃T› have {x,y}∈Pow(⋃T) by auto
    moreover
    note AS
    ultimately have {x,y}{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V})) unfolding antiProperty_def
      by simp
    then have {x,y}≲1 using linord_spectrum by auto
    moreover
    have x∈{x,y} by auto
    ultimately have {x}={x,y} using lepoll_1_is_sing[of {x,y}x] by auto
    moreover
    have y∈{x,y} by auto
    ultimately
    have y∈{x} by auto
    then have x=y by auto
    then have False using ‹x≠y› by auto
  }
  then show T {is T₁} by auto
next
  assume T1:T {is T₁}
  {
    fix A
```

```
    assume A_def:A∈Pow(⋃T)IsLinOrder((T{restricted to}A) ,{⟨U,V⟩∈Pow(⋃(T{restricted
to}A))×Pow(⋃(T{restricted to}A)). U⊆V})
    {
      fix x
      assume AS1:x∈A
      {
        fix y
        assume AS:y∈Ax≠y
        with AS1 have {x,y}∈Pow(⋃T) using ‹A∈Pow(⋃T)› by auto
        from ‹x∈A›‹y∈A› have {x,y}∈Pow(A) by auto
        from ‹{x,y}∈Pow(⋃T)› have T11:(T{restricted to}{x,y}){is T₁}
using T1_here T1 by auto
        moreover
        have tot:⋃(T{restricted to}{x,y})={x,y} unfolding RestrictedTo_def
using ‹{x,y}∈Pow(⋃T)› by auto
        moreover
        note AS(2)
        ultimately obtain U where x∈Uy∉UU∈(T{restricted to}{x,y}) un-
folding isT1_def by auto
        moreover
        from AS(2) tot T11 obtain V where y∈Vx∉VV∈(T{restricted to}{x,y})
unfolding isT1_def by auto
        ultimately have x∈U-Vy∈V-UU∈(T{restricted to}{x,y})V∈(T{restricted
to}{x,y}) by auto
        then have ¬(U⊆V∨V⊆U)U∈(T{restricted to}{x,y})V∈(T{restricted
to}{x,y}) by auto
        then have ¬({⟨U,V⟩∈Pow(⋃(T{restricted to}{x,y}))×Pow(⋃(T{restricted
to}{x,y})). U⊆V} {is total on} (T{restricted to}{x,y}))
          unfolding IsTotal_def by auto
        then have ¬(IsLinOrder((T{restricted to}{x,y}),{⟨U,V⟩∈Pow(⋃(T{restricted
to}{x,y}))×Pow(⋃(T{restricted to}{x,y})). U⊆V}))
          unfolding IsLinOrder_def by auto
        moreover
        {
          have (T{restricted to}A) {is a topology} using Top_1_L4 by
auto
          moreover
          note A_def(2) linordtop_here
          ultimately have ∀B∈Pow(⋃(T{restricted to}A)). IsLinOrder((T{restricted
to}A){restricted to}B ,{⟨U,V⟩∈Pow(⋃((T{restricted to}A){restricted to}B))×Pow(⋃((T{restri
to}A){restricted to}B)). U⊆V})
            unfolding IsHer_def by auto
          moreover
          have tot:⋃(T{restricted to}A)=A unfolding RestrictedTo_def
using ‹A∈Pow(⋃T)› by auto
          ultimately have  ∀B∈Pow(A). IsLinOrder((T{restricted to}A){restricted
to}B ,{⟨U,V⟩∈Pow(⋃((T{restricted to}A){restricted to}B))×Pow(⋃((T{restricted
to}A){restricted to}B)). U⊆V}) by auto
          moreover
```

```
          have ∀B∈Pow(A). (T{restricted to}A){restricted to}B=T{restricted
to}B using subspace_of_subspace ‹A∈Pow(⋃T)› by auto
          ultimately
          have ∀B∈Pow(A). IsLinOrder((T{restricted to}B) ,{⟨U,V⟩∈Pow(⋃((T{restricted
to}A){restricted to}B))×Pow(⋃((T{restricted to}A){restricted to}B)).
U⊆V}) by auto
          moreover
          have ∀B∈Pow(A). ⋃((T{restricted to}A){restricted to}B)=B us-
ing ‹A∈Pow(⋃T)› unfolding RestrictedTo_def by auto
          ultimately have ∀B∈Pow(A). IsLinOrder((T{restricted to}B) ,{⟨U,V⟩∈Pow(B)×Pow(B).
U⊆V}) by auto
          with ‹{x,y}∈Pow(A)› have IsLinOrder((T{restricted to}{x,y})
,{⟨U,V⟩∈Pow({x,y})×Pow({x,y}). U⊆V}) by auto
        }
        ultimately have False using tot by auto
      }
      then have A={x} using AS1 by auto
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
      then have A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V})) using linord_spectrum
        by auto
    }
    moreover
    {
      assume A=0
      then have A≈0 by auto
      moreover
      have 0≲1 using empty_lepollI by auto
      ultimately have A≲1 using eq_lepoll_trans by auto
      then have A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V})) using linord_spectrum
        by auto
    }
    ultimately have A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V})) by blast
  }
  then show T{is anti-}(λT. IsLinOrder(T, {⟨U,V⟩ ∈ Pow(⋃T) × Pow(⋃T)
. U ⊆ V})) unfolding antiProperty_def
    by auto
qed
```

In conclusion, T_1 is also an anti-property.

Let's define some anti-properties that we'll use in the future.

```
definition
  IsAntiComp (_{is anti-compact})
  where T{is anti-compact} ≡ T{is anti-}(λT. (⋃T){is compact in}T)

definition
```

```
  IsAntiLin (_{is anti-lindeloef})
  where T{is anti-lindeloef} ≡ T{is anti-}(λT. ((⋃T){is lindeloef in}T))
```

Anti-compact spaces are also called pseudo-finite spaces in literature before the concept of anti-property was defined.

end

82 Topology 6

theory Topology_ZF_6 **imports** Topology_ZF_4 Topology_ZF_2 Topology_ZF_1

begin

This theory deals with the relations between continuous functions and convergence of filters. At the end of the file there some results about the building of functions in cartesian products.

82.1 Image filter

First of all, we will define the appropriate tools to work with functions and filters together.

We define the image filter as the collections of supersets of of images of sets from a filter.

```
definition
  ImageFilter (_[_].._ 98)
  where 𝔉 {is a filter on} X ⟹ f:X→Y ⟹ f[𝔉]..Y ≡ {A∈Pow(Y). ∃D∈{f(B)
.B∈𝔉}. D⊆A}
```

Note that in the previous definition, it is necessary to state Y as the final set because f is also a function to every superset of its range. X can be changed by domain(f) without any change in the definition.

```
lemma base_image_filter:
  assumes 𝔉 {is a filter on} X f:X→Y
  shows {fB .B∈𝔉} {is a base filter} (f[𝔉]..Y) and (f[𝔉]..Y) {is a filter
on} Y
proof-
  {
    assume 0 ∈ {fB .B∈𝔉}
    then obtain B where B∈𝔉 and f_B:fB=0 by auto
    then have B∈Pow(X) using assms(1) IsFilter_def by auto
    then have fB={fb. b∈B} using image_fun assms(2) by auto
    with f_B have {fb. b∈B}=0 by auto
    then have B=0 by auto
    with ‹B∈𝔉› have False using IsFilter_def assms(1) by auto
  }
  then have 0∉{fB .B∈𝔉} by auto
```

```
  moreover
  from assms(1) obtain S where S∈𝔉 using IsFilter_def by auto
  then have fS∈{fB .B∈𝔉} by auto
  then have nA:{fB .B∈𝔉}≠0 by auto
  moreover
  {
    fix A B
    assume A∈{fB .B∈𝔉} and B∈{fB .B∈𝔉}
    then obtain AB BB where A=fAB B=fBB AB∈𝔉 BB∈𝔉 by auto
    then have A∩B=(fAB)∩(fBB) by auto
    then have I: f(AB∩BB)⊆A∩B by auto
    moreover
    from assms(1) I ‹AB∈𝔉›‹BB∈𝔉› have AB∩BB∈𝔉 using IsFilter_def by
auto
    ultimately have ∃D∈{fB .B∈𝔉}. D⊆A∩B by auto
  }
  then have ∀A∈{fB .B∈𝔉}. ∀B∈{fB .B∈𝔉}. ∃D∈{fB .B∈𝔉}. D⊆A∩B by auto
  ultimately have sbc:{fB .B∈𝔉} {satisfies the filter base condition}

    using SatisfiesFilterBase_def by auto
  moreover
  {
    fix t
    assume t∈{fB . B∈𝔉}
    then obtain B where B∈𝔉 and im_def:fB=t by auto
    with assms(1) have B∈Pow(X) unfolding IsFilter_def by auto
    with im_def assms(2) have t={fx. x∈B} using image_fun by auto
    with assms(2) ‹B∈Pow(X)› have t⊆Y using apply_funtype by auto
    }
  then have nB:{fB . B∈𝔉}⊆Pow(Y) by auto
  ultimately
  have (({fB .B∈𝔉} {is a base filter} {A ∈ Pow(Y) . ∃D∈{fB .B∈𝔉}. D
⊆ A}) ∧ (⋃{A ∈ Pow(Y) . ∃D∈{fB .B∈𝔉}. D ⊆ A}=Y)) using base_unique_filter_set2

    by force
  then have {fB .B∈𝔉} {is a base filter} {A ∈ Pow(Y) . ∃D∈{fB .B∈𝔉}.
D ⊆ A} by auto
  with assms show {fB .B∈𝔉} {is a base filter} (f[𝔉]..Y) using ImageFilter_def
by auto
  moreover
  note sbc
  moreover
  {
    from nA obtain D where I: D∈{fB .B∈𝔉} by blast
    moreover from I nB have D⊆Y by auto
    ultimately have Y∈{A∈Pow(Y). ∃D∈{fB .B∈𝔉}. D⊆A} by auto
  }
  then have ⋃{A∈Pow(Y). ∃D∈{fB .B∈𝔉}. D⊆A}=Y by auto
  ultimately show (f[𝔉]..Y) {is a filter on} Y using basic_filter
```

```
    ImageFilter_def assms by auto
qed
```

82.2 Continuous at a point vs. globally continuous

In this section we show that continuity of a function implies local continuity (at a point) and that local continuity at all points implies (global) continuity.

If a function is continuous, then it is continuous at every point.

```
lemma cont_global_imp_continuous_x:
  assumes x∈⋃τ₁ IsContinuous(τ₁,τ₂,f) f:(⋃τ₁)→(⋃τ₂) x∈⋃τ₁
  shows ∀U∈τ₂. f(x)∈U ⟶ (∃V∈τ₁. x∈V ∧ f(V)⊆U)
proof-
  {
    fix U
    assume AS:U∈τ₂ f(x)∈U
    then have f-(U)∈τ₁ using assms(2) IsContinuous_def by auto
    moreover
    from assms(3) have f(f-(U))⊆U using function_image_vimage fun_is_fun

      by auto
    moreover
    from assms(3) assms(4) AS(2) have x∈f-(U) using func1_1_L15 by auto
    ultimately have ∃V∈τ₁. x∈V ∧ fV⊆U by auto
  }
  then show ∀U∈τ₂. f(x)∈U ⟶ (∃V∈τ₁. x∈V ∧ f(V)⊆U) by auto
qed
```

A function that is continuous at every point of its domain is continuous.

```
lemma ccontinuous_all_x_imp_cont_global:
  assumes ∀x∈⋃τ₁. ∀U∈τ₂. fx∈U ⟶ (∃V∈τ₁. x∈V ∧ fV⊆U) f∈(⋃τ₁)→(⋃τ₂)
and
    τ₁ {is a topology}
  shows IsContinuous(τ₁,τ₂,f)
proof-
  {
    fix U
    assume U∈τ₂
    {
      fix x
      assume AS: x∈f-U
      note ‹U∈τ₂›
      moreover
      from assms(2) have f - U ⊆ ⋃τ₁ using func1_1_L6A by auto
      with AS have x∈⋃τ₁ by auto
      with assms(1) have ∀U∈τ₂. fx∈U ⟶ (∃V∈τ₁. x∈V ∧ fV⊆U) by auto
      moreover
      from AS assms(2) have fx∈U using func1_1_L15 by auto
      ultimately have ∃V∈τ₁. x∈V ∧ fV⊆U by auto
```

```
      then obtain V where I: V∈τ₁ x∈V f(V)⊆U by auto
      moreover
      from I have V⊆⋃τ₁ by auto
      moreover
      from assms(2) ‹V⊆⋃τ₁› have V⊆f-(fV) using func1_1_L9 by auto
      ultimately have V ⊆ f-(U) by blast
      with ‹V∈τ₁› ‹x∈V› have ∃V∈τ₁. x∈V ∧ V ⊆ f-(U) by auto
    } hence ∀x∈f-(U). ∃V∈τ₁. x∈V ∧ V⊆f-(U) by auto
    with assms(3) have f-(U) ∈ τ₁ using topology0.open_neigh_open topology0_def

      by auto
  }
  hence ∀U∈τ₂. f-U∈τ₁ by auto
  then show thesis using IsContinuous_def by auto
qed
```

82.3 Continuous functions and filters

In this section we consider the relations between filters and continuity.

If the function is continuous then if the filter converges to a point the image filter converges to the image point.

```
lemma (in two_top_spaces0) cont_imp_filter_conver_preserved:
  assumes 𝔉 {is a filter on} X₁ f {is continuous} 𝔉 →F x {in} τ₁
  shows (f[𝔉]..X₂) →F (f(x)) {in} τ₂
proof -
  from assms(1) assms(3) have x∈X₁
    using topology0.FilterConverges_def topol_cntxs_valid(1) X1_def by
auto
  have topology0(τ₂) using topol_cntxs_valid(2) by simp
  moreover from assms(1) have (f[𝔉]..X₂) {is a filter on} (⋃τ₂) and
{fB .B∈𝔉} {is a base filter} (f[𝔉]..X₂)
    using base_image_filter fmapAssum X1_def X2_def by auto
  moreover have ∀U∈Pow(⋃τ₂). (fx)∈Interior(U,τ₂) ⟶ (∃D∈{fB .B∈𝔉}.
D⊆U)
  proof -
    { fix U
    assume U∈Pow(X₂) (fx)∈Interior(U,τ₂)
    with ‹x∈X₁› have xim: x∈f-(Interior(U,τ₂)) and sub: f-(Interior(U,τ₂))∈Pow(X₁)

      using func1_1_L6A fmapAssum func1_1_L15 fmapAssum by auto
    note sub
    moreover
    have Interior(U,τ₂)∈τ₂ using topology0.Top_2_L2 topol_cntxs_valid(2)
by auto
    with assms(2) have f-(Interior(U,τ₂))∈τ₁ unfolding isContinuous_def
IsContinuous_def
      by auto
    with xim have x∈Interior(f-(Interior(U,τ₂)),τ₁)
```

```
      using topology0.Top_2_L3 topol_cntxs_valid(1) by auto
    moreover from assms(1) assms(3) have {U∈Pow(X1). x∈Interior(U,τ1)}⊆𝔉
        using topology0.FilterConverges_def topol_cntxs_valid(1) X1_def by auto
    ultimately have f-(Interior(U,τ2))∈𝔉 by auto
    moreover have f(f-(Interior(U,τ2)))⊆Interior(U,τ2)
      using function_image_vimage fun_is_fun fmapAssum by auto
    then have f(f-(Interior(U,τ2)))⊆U
      using topology0.Top_2_L1 topol_cntxs_valid(2) by auto
    ultimately have ∃D∈{f(B) .B∈𝔉}. D⊆U by auto
    } thus thesis by auto
  qed
  moreover from fmapAssum ‹x∈X1›  have f(x) ∈ X2
    by (rule apply_funtype)
  hence f(x) ∈ ⋃τ2 by simp
  ultimately show thesis by (rule topology0.convergence_filter_base2)
qed
```

Continuity in filter at every point of the domain implies global continuity.

```
lemma (in two_top_spaces0) filter_conver_preserved_imp_cont:
  assumes ∀x∈⋃τ1. ∀𝔉. ((𝔉 {is a filter on} X1) ∧ (𝔉 →F x {in} τ1)) ⟶ ((f[𝔉]..X2) →F (fx) {in} τ2)
  shows f{is continuous}
proof-
  {
    fix x
    assume as2: x∈⋃τ1
    with assms have reg:
      ∀𝔉. ((𝔉 {is a filter on} X1) ∧ (𝔉 →F x {in} τ1)) ⟶ ((f[𝔉]..X2) →F (fx) {in} τ2)
      by auto
    let Neig = {U ∈ Pow(⋃τ1) . x ∈ Interior(U, τ1)}
    from as2 have NFil: Neig{is a filter on}X1 and NCon: Neig →F x {in} τ1
      using topol_cntxs_valid(1) topology0.neigh_filter by auto
    {
      fix U
      assume U∈τ2 fx∈U
      then have U∈Pow(⋃τ2) fx∈Interior(U,τ2) using topol_cntxs_valid(2) topology0.Top_2_L3 by auto
      moreover
      from NCon NFil reg have (f[Neig]..X2) →F (fx) {in} τ2 by auto

      moreover have (f[Neig]..X2) {is a filter on} X2
        using base_image_filter(2) NFil fmapAssum by auto
      ultimately have U∈(f[Neig]..X2)
        using topology0.FilterConverges_def topol_cntxs_valid(2) unfold-
```

```
ing X1_def X2_def
        by auto
      moreover
      from fmapAssum NFil have {fB .B∈Neig} {is a base filter} (f[Neig]..X₂)

        using base_image_filter(1) X1_def X2_def by auto
      ultimately have ∃V∈{fB .B∈Neig}. V⊆U using basic_element_filter
by blast
      then obtain B where B∈Neig fB⊆U by auto
      moreover
      have Interior(B,τ₁)⊆B using topology0.Top_2_L1 topol_cntxs_valid(1)
by auto
      hence fInterior(B,τ₁) ⊆ f(B) by auto
      moreover have Interior(B,τ₁)∈τ₁
        using topology0.Top_2_L2 topol_cntxs_valid(1) by auto
      ultimately have ∃V∈τ₁. x∈V ∧ fV⊆U by force
    }
    hence ∀U∈τ₂. fx∈U ⟶ (∃V∈τ₁. x∈V ∧ fV⊆U) by auto
  }
  hence ∀x∈⋃τ₁. ∀U∈τ₂. fx∈U ⟶ (∃V∈τ₁. x∈V ∧ fV⊆U) by auto
  then show thesis
    using ccontinuous_all_x_imp_cont_global fmapAssum X1_def X2_def isContinuous_def
tau1_is_top
    by auto
qed

end
```

83 Topology 7

```
theory Topology_ZF_7 imports Topology_ZF_5
begin
```

83.1 Connection Properties

Another type of topological properties are the connection properties. These properties establish if the space is formed of several pieces or just one.

A space is connected iff there is no clopen set other that the empty set and the total set.

```
definition IsConnected (_{is connected} 70)
  where T {is connected} ≡ ∀U. (U∈T ∧ (U {is closed in}T)) ⟶ U=0∨U=⋃T

lemma indiscrete_connected:
  shows {0,X} {is connected}
  unfolding IsConnected_def IsClosed_def by auto
```

The anti-property of connectedness is called total-diconnectedness.

```
definition IsTotDis (_ {is totally-disconnected} 70)
  where IsTotDis ≡ ANTI(IsConnected)

lemma conn_spectrum:
  shows (A{is in the spectrum of}IsConnected) ⟷ A≲1
proof
  assume A{is in the spectrum of}IsConnected
  then have ∀T. (T{is a topology}∧⋃T≈A) ⟶ (T{is connected}) using
Spec_def by auto
  moreover
  have Pow(A){is a topology} using Pow_is_top by auto
  moreover
  have ⋃(Pow(A))=A by auto
  then have ⋃(Pow(A))≈A by auto
  ultimately have Pow(A) {is connected} by auto
  {
    assume A≠0
    then obtain E where E∈A by blast
    then have {E}∈Pow(A) by auto
    moreover
    have A-{E}∈Pow(A) by auto
    ultimately have {E}∈Pow(A)∧{E}{is closed in}Pow(A) unfolding IsClosed_def
by auto
    with ‹Pow(A) {is connected}› have {E}=A unfolding IsConnected_def
by auto
    then have A≈1 using singleton_eqpoll_1 by auto
    then have A≲1 using eqpoll_imp_lepoll by auto
  }
  moreover
  {
    assume A=0
    then have A≲1 using empty_lepollI[of 1] by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T{is a topology}⋃T≈A
    {
      assume ⋃T=0
      with ‹T{is a topology}› have T={0} using empty_open by auto
      then have T{is connected} unfolding IsConnected_def by auto
    }
    moreover
    {
      assume ⋃T≠0
      moreover
      from ‹A≲1›‹⋃T≈A› have ⋃T≲1 using eq_lepoll_trans by auto
```

```
      ultimately
      obtain E where ⋃T={E} using lepoll_1_is_sing by blast
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow({E}) by auto
      then have T⊆{0,{E}} by blast
      with ‹T{is a topology}› have {0}⊆T T⊆{0,{E}} using empty_open by
auto
      then have T{is connected} unfolding IsConnected_def by auto
    }
    ultimately have T{is connected} by auto
  }
  then show A{is in the spectrum of}IsConnected unfolding Spec_def by
auto
qed
```

The discrete space is a first example of totally-disconnected space.

```
lemma discrete_tot_dis:
  shows Pow(X) {is totally-disconnected}
proof-
  {
    fix A assume A∈Pow(X) and con:(Pow(X){restricted to}A){is connected}
    have res:(Pow(X){restricted to}A)=Pow(A) unfolding RestrictedTo_def
using ‹A∈Pow(X)›
      by blast
    {
      assume A=0
      then have A≲1 using empty_lepollI[of 1] by auto
      then have A{is in the spectrum of}IsConnected using conn_spectrum
by auto
    }
    moreover
    {
      assume A≠0
      then obtain E where E∈A by blast
      then have {E}∈Pow(A) by auto
      moreover
      have A-{E}∈Pow(A) by auto
      ultimately have {E}∈Pow(A)∧{E}{is closed in}Pow(A) unfolding IsClosed_def
by auto
      with con res have {E}=A unfolding IsConnected_def by auto
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
      then have A{is in the spectrum of}IsConnected using conn_spectrum
by auto
    }
    ultimately have A{is in the spectrum of}IsConnected by auto
  }
  then show thesis unfolding IsTotDis_def antiProperty_def by auto
```

```
qed
```

An space is hyperconnected iff every two non-empty open sets meet.

```
definition IsHConnected (_{is hyperconnected}90)
  where T{is hyperconnected} ≡∀U V. U∈T ∧ V∈T ∧ U∩V=0 ⟶ U=0∨V=0
```

Every hyperconnected space is connected.

```
lemma HConn_imp_Conn:
  assumes T{is hyperconnected}
  shows T{is connected}
proof-
  {
    fix U
    assume U∈TU {is closed in}T
    then have ⋃T-U∈TU∈T using IsClosed_def by auto
    moreover
    have (⋃T-U)∩U=0 by auto
    moreover
    note assms
    ultimately
    have U=0∨(⋃T-U)=0 using IsHConnected_def by auto
    with ‹U∈T› have U=0∨U=⋃T by auto
  }
  then show thesis using IsConnected_def by auto
qed

lemma Indiscrete_HConn:
  shows {0,X}{is hyperconnected}
  unfolding IsHConnected_def by auto
```

A first example of an hyperconnected space but not indiscrete, is the cofinite topology on the natural numbers.

```
lemma Cofinite_nat_HConn:
  assumes ¬(X≺nat)
  shows (CoFinite X){is hyperconnected}
proof-
  {
    fix U V
    assume U∈(CoFinite X)V∈(CoFinite X)U∩V=0
    then have eq:(X-U)≺nat∨U=0(X-V)≺nat∨V=0 unfolding Cofinite_def
      CoCardinal_def by auto
    from ‹U∩V=0› have un:(X-U)∪(X-V)=X by auto
    {
      assume AS:(X-U)≺nat(X-V)≺nat
      from un have X≺nat using less_less_imp_un_less[OF AS InfCard_nat]
by auto
      then have False using assms by auto
    }
    with eq(1,2) have U=0∨V=0 by auto
```

```
  }
  then show (CoFinite X){is hyperconnected} using IsHConnected_def by
auto
qed

lemma HConn_spectrum:
  shows (A{is in the spectrum of}IsHConnected) ⟷ A≲1
proof
  assume A{is in the spectrum of}IsHConnected
  then have ∀T. (T{is a topology}∧⋃T≈A) ⟶ (T{is hyperconnected})
using Spec_def by auto
  moreover
  have Pow(A){is a topology} using Pow_is_top by auto
  moreover
  have ⋃(Pow(A))=A by auto
  then have ⋃(Pow(A))≈A by auto
  ultimately
  have HC_Pow:Pow(A){is hyperconnected} by auto
  {
    assume A=0
    then have A≲1 using empty_lepollI by auto
  }
  moreover
  {
    assume A≠0
    then obtain e where e∈A by blast
    then have {e}∈Pow(A) by auto
    moreover
    have A-{e}∈Pow(A) by auto
    moreover
    have {e}∩(A-{e})=0 by auto
    moreover
    note HC_Pow
    ultimately have A-{e}=0 unfolding IsHConnected_def by blast
    with ‹e∈A› have A={e} by auto
    then have A≈1 using singleton_eqpoll_1 by auto
    then have A≲1 using eqpoll_imp_lepoll by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T{is a topology}⋃T≈A
    {
      assume ⋃T=0
      with ‹T{is a topology}› have T={0} using empty_open by auto
      then have T{is hyperconnected} unfolding IsHConnected_def by auto
    }
```

```
    moreover
    {
      assume ⋃T≠0
      moreover
      from ‹A≲1›‹⋃T≈A› have ⋃T≲1 using eq_lepoll_trans by auto
      ultimately
      obtain E where ⋃T={E} using lepoll_1_is_sing by blast
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow({E}) by auto
      then have T⊆{0,{E}} by blast
      with ‹T{is a topology}› have {0}⊆T T⊆{0,{E}} using empty_open by
auto
      then have T{is hyperconnected} unfolding IsHConnected_def by auto
    }
    ultimately have T{is hyperconnected} by auto
  }
  then show A{is in the spectrum of}IsHConnected unfolding Spec_def by
auto
qed
```

In the following results we will show that anti-hyperconnectedness is a separation property between T_1 and T_2. We will show also that both implications are proper.

First, the closure of a point in every topological space is always hyperconnected. This is the reason why every anti-hyperconnected space must be T_1: every singleton must be closed.

```
lemma (in topology0)cl_point_imp_HConn:
  assumes x∈⋃T
  shows (T{restricted to}Closure({x},T)){is hyperconnected}
proof-
  from assms have sub:Closure({x},T)⊆⋃T using Top_3_L11 by auto
  then have tot:⋃(T{restricted to}Closure({x},T))=Closure({x},T) un-
folding RestrictedTo_def by auto
  {
    fix A B
    assume AS:A∈(T{restricted to}Closure({x},T))B∈(T{restricted to}Closure({x},T))A∩B=0
    then have B⊆⋃((T{restricted to}Closure({x},T)))A⊆⋃((T{restricted
to}Closure({x},T)))
      by auto
    with tot have B⊆Closure({x},T)A⊆Closure({x},T) by auto
    from AS(1,2) obtain UA UB where UAUB:UA∈TUB∈TA=UA∩Closure({x},T)B=UB∩Closure({x},T)
      unfolding RestrictedTo_def by auto
    then have Closure({x},T)-A=Closure({x},T)-(UA∩Closure({x},T)) Closure({x},T)-B=Closure
      by auto
    then have Closure({x},T)-A=Closure({x},T)-(UA) Closure({x},T)-B=Closure({x},T)-(UB)
      by auto
    with sub have Closure({x},T)-A=Closure({x},T)∩(⋃T-UA) Closure({x},T)-B=Closure({x},T)∩
```

```
by auto
    moreover
    from UAUB have (⋃T-UA){is closed in}T(⋃T-UB){is closed in}T us-
ing Top_3_L9 by auto
    moreover
    have Closure({x},T){is closed in}T using cl_is_closed assms by auto
    ultimately have (Closure({x},T)-A){is closed in}T(Closure({x},T)-B){is
closed in}T
      using Top_3_L5(1) by auto
    moreover
    {
      have x∈Closure({x},T) using cl_contains_set assms by auto
      moreover
      from AS(3) have x∉A∨x∉B by auto
      ultimately have x∈(Closure({x},T)-A)∨x∈(Closure({x},T)-B) by auto
    }
    ultimately have Closure({x},T)⊆(Closure({x},T)-A) ∨ Closure({x},T)⊆(Closure({x},T)-B)
      using Top_3_L13 by auto
    then have A∩Closure({x},T)=0 ∨ B∩Closure({x},T)=0 by auto
    with ‹B⊆Closure({x},T)›‹A⊆Closure({x},T)› have A=0∨B=0 using cl_contains_set
assms by blast
  }
  then show thesis unfolding IsHConnected_def by auto
qed
```

A consequence is that every totally-disconnected space is T_1.

```
lemma (in topology0) tot_dis_imp_T1:
  assumes T{is totally-disconnected}
  shows T{is T1}
proof-
  {
    fix x y
    assume y∈⋃Tx∈⋃Ty≠x
    then have (T{restricted to}Closure({x},T)){is hyperconnected} us-
ing cl_point_imp_HConn by auto
    then have (T{restricted to}Closure({x},T)){is connected} using HConn_imp_Conn
by auto
    moreover
    from ‹x∈⋃T› have Closure({x},T)⊆⋃T using Top_3_L11(1) by auto
    moreover
    note assms
    ultimately have Closure({x},T){is in the spectrum of}IsConnected un-
folding IsTotDis_def antiProperty_def
      by auto
    then have Closure({x},T)≲1 using conn_spectrum by auto
    moreover
    from ‹x∈⋃T› have x∈Closure({x},T) using cl_contains_set by auto
    ultimately have Closure({x},T)={x} using lepoll_1_is_sing[of Closure({x},T)
x] by auto
```

```
    then have {x}{is closed in}T using Top_3_L8 ‹x∈⋃T› by auto
    then have ⋃T-{x}∈T unfolding IsClosed_def by auto
    moreover
    from ‹y∈⋃T›‹y≠x› have y∈⋃T-{x}∧x∉⋃T-{x} by auto
    ultimately have ∃U∈T. y∈U∧x∉U by force
  }
  then show thesis unfolding isT1_def by auto
qed
```

In the literature, there exists a class of spaces called sober spaces; where the only non-empty closed hyperconnected subspaces are the closures of points and closures of different singletons are different.

```
definition IsSober (_{is sober}90)
  where T{is sober} ≡ ∀A∈Pow(⋃T)-{0}. (A{is closed in}T ∧ ((T{restricted
to}A){is hyperconnected})) ⟶ (∃x∈⋃T. A=Closure({x},T) ∧ (∀y∈⋃T. A=Closure({y},T)
⟶ y=x) )
```

Being sober is weaker than being anti-hyperconnected.

```
theorem (in topology0) anti_HConn_imp_sober:
  assumes T{is anti-}IsHConnected
  shows T{is sober}
proof-
  {
    fix A assume A∈Pow(⋃T)-{0}A{is closed in}T(T{restricted to}A){is
hyperconnected}
    with assms have A{is in the spectrum of}IsHConnected unfolding antiProperty_def
by auto
    then have A≲1 using HConn_spectrum by auto
    moreover
    with ‹A∈Pow(⋃T)-{0}› have A≠0 by auto
    then obtain x where x∈A by auto
    ultimately have A={x} using lepoll_1_is_sing by auto
    with ‹A{is closed in}T› have {x}{is closed in}T by auto
    moreover from ‹x∈A› ‹A∈Pow(⋃T)-{0}› have {x}∈Pow(⋃T) by auto
    ultimately
    have Closure({x},T)={x} unfolding Closure_def ClosedCovers_def by
auto
    with ‹A={x}› have A=Closure({x},T) by auto
    moreover
    {
      fix y assume y∈⋃TA=Closure({y},T)
      then have {y}⊆Closure({y},T) using cl_contains_set by auto
      with ‹A=Closure({y},T)› have y∈A by auto
      with ‹A={x}› have y=x by auto
    }
    then have ∀y∈⋃T. A=Closure({y},T) ⟶ y=x by auto
    moreover note ‹{x}∈Pow(⋃T)›
    ultimately have ∃x∈⋃T. A=Closure({x},T)∧(∀y∈⋃T. A=Closure({y},T)
⟶ y=x) by auto
```

```
  }
  then show thesis using IsSober_def by auto
qed
```

Every sober space is T_0.

```
lemma (in topology0) sober_imp_T0:
  assumes T{is sober}
  shows T{is T0}
proof-
  {
    fix x y
    assume AS:x∈⋃Ty∈⋃Tx≠y∀U∈T. x∈U ⟷ y∈U
    from ‹x∈⋃T› have clx:Closure({x},T) {is closed in}T using cl_is_closed
by auto
    with ‹x∈⋃T› have (⋃T-Closure({x},T))∈T using Top_3_L11(1) unfold-
ing IsClosed_def by auto
    moreover
    from ‹x∈⋃T› have x∈Closure({x},T) using cl_contains_set by auto
    moreover
    note AS(1,4)
    ultimately have y∉(⋃T-Closure({x},T)) by auto
    with AS(2) have y∈Closure({x},T) by auto
    with clx have ineq1:Closure({y},T)⊆Closure({x},T) using Top_3_L13
by auto
    from ‹y∈⋃T› have cly:Closure({y},T) {is closed in}T using cl_is_closed
by auto
    with ‹y∈⋃T› have (⋃T-Closure({y},T))∈T using Top_3_L11(1) unfold-
ing IsClosed_def by auto
    moreover
    from ‹y∈⋃T› have y∈Closure({y},T) using cl_contains_set by auto
    moreover
    note AS(2,4)
    ultimately have x∉(⋃T-Closure({y},T)) by auto
    with AS(1) have x∈Closure({y},T) by auto
    with cly have Closure({x},T)⊆Closure({y},T) using Top_3_L13 by auto
    with ineq1 have eq:Closure({x},T)=Closure({y},T) by auto
    have Closure({x},T)∈Pow(⋃T)-{0} using Top_3_L11(1) ‹x∈⋃T› ‹x∈Closure({x},T)›
by auto
    moreover note assms clx
    ultimately have ∃t∈⋃T.( Closure({x},T) = Closure({t}, T) ∧ (∀y∈⋃T.
Closure({x},T) = Closure({y}, T) ⟶ y = t))
      unfolding IsSober_def using cl_point_imp_HConn[OF ‹x∈⋃T›] by auto
    then obtain t where t_def:t∈⋃TClosure({x},T) = Closure({t}, T)∀y∈⋃T.
Closure({x},T) = Closure({y}, T) ⟶ y = t
      by blast
    with eq have y=t using ‹y∈⋃T› by auto
    moreover from t_def ‹x∈⋃T› have x=t by blast
    ultimately have y=x by auto
    with ‹x≠y› have False by auto
```

```
  }
  then have ∀x y. x∈⋃T∧y∈⋃T∧x≠y ⟶ (∃U∈T. (x∈U∧y∉U)∨(y∈U∧x∉U))
by auto
  then show thesis using isT0_def by auto
qed
```

Every T_2 space is anti-hyperconnected.

```
theorem (in topology0) T2_imp_anti_HConn:
  assumes T{is T₂}
  shows T{is anti-}IsHConnected
proof-
  {
    fix TT
    assume TT{is a topology} TT{is hyperconnected}TT{is T₂}
    {
      assume ⋃TT=0
      then have ⋃TT≲1 using empty_lepollI by auto
      then have (⋃TT){is in the spectrum of}IsHConnected using HConn_spectrum
by auto
    }
    moreover
    {
      assume ⋃TT≠0
      then obtain x where x∈⋃TT by blast
      {
        fix y
        assume y∈⋃TTx≠y
        with ‹TT{is T₂}›‹x∈⋃TT› obtain U V where U∈TTV∈TTx∈Uy∈VU∩V=0
unfolding isT2_def by blast
        with ‹TT{is hyperconnected}› have False using IsHConnected_def
by auto
      }
      with ‹x∈⋃TT› have ⋃TT={x} by auto
      then have ⋃TT≈1 using singleton_eqpoll_1 by auto
      then have ⋃TT≲1 using eqpoll_imp_lepoll by auto
      then have (⋃TT){is in the spectrum of}IsHConnected using HConn_spectrum
by auto
    }
    ultimately have (⋃TT){is in the spectrum of}IsHConnected by blast
  }
  then have ∀T. ((T{is a topology}∧(T{is hyperconnected})∧(T{is T₂}))⟶
((⋃T){is in the spectrum of}IsHConnected))
    by auto
  moreover
  note here_T2
  ultimately
  have ∀T.  T{is a topology} ⟶ ((T{is T₂})⟶(T{is anti-}IsHConnected))
using Q_P_imp_Spec[where P=IsHConnected and Q=isT2]
    by auto
```

```
  then show thesis using assms topSpaceAssum by auto
qed
```

Every anti-hyperconnected space is T_1.

```
theorem anti_HConn_imp_T1:
  assumes T{is anti-}IsHConnected
  shows T{is T₁}
proof-
  {
    fix x y
    assume x∈⋃Ty∈⋃Tx≠y
    {
      assume AS:∀U∈T. x∉U∨y∈U
      from ‹x∈⋃T›‹y∈⋃T› have {x,y}∈Pow(⋃T) by auto
      then have sub:(T{restricted to}{x,y})⊆Pow({x,y}) using RestrictedTo_def
by auto
      {
        fix U V
        assume H:U∈T{restricted to}{x,y} V∈(T{restricted to}{x,y})U∩V=0
        with AS have x∈U⟶y∈Ux∈V⟶y∈V unfolding RestrictedTo_def by
auto
        with H(1,2) sub have x∈U⟶U={x,y}x∈V⟶V={x,y} by auto
        with H sub have x∈U⟶(U={x,y}∧V=0)x∈V⟶(V={x,y}∧U=0) by auto
        then have (x∈U∨x∈V)⟶(U=0∨V=0) by auto
        moreover
        from sub H have (x∉U∧x∉V)⟶ (U=0∨V=0) by blast
        ultimately have U=0∨V=0 by auto
      }
      then have (T{restricted to}{x,y}){is hyperconnected} unfolding IsHConnected_def
by auto
      with assms‹{x,y}∈Pow(⋃T)› have {x,y}{is in the spectrum of}IsHConnected
unfolding antiProperty_def
        by auto
      then have {x,y}≲1 using HConn_spectrum by auto
      moreover
      have x∈{x,y} by auto
      ultimately have {x,y}={x} using lepoll_1_is_sing[of {x,y}x] by auto
      moreover
      have y∈{x,y} by auto
      ultimately have y∈{x} by auto
      then have y=x by auto
      with ‹x≠y› have False by auto
    }
    then have ∃U∈T. x∈U∧y∉U by auto
  }
  then show thesis using isT1_def by auto
qed
```

There is at least one topological space that is T_1, but not anti-hyperconnected.

This space is the cofinite topology on the natural numbers.

```
lemma Cofinite_not_anti_HConn:
  shows ¬((CoFinite nat){is anti-}IsHConnected) and (CoFinite nat){is
T₁}
proof-
  {
    assume (CoFinite nat){is anti-}IsHConnected
    moreover
    have ⋃(CoFinite nat)=nat unfolding Cofinite_def using union_cocardinal
by auto
    moreover
    have (CoFinite nat){restricted to}nat=(CoFinite nat) using subspace_cocardinal
unfolding Cofinite_def
      by auto
    moreover
    have ¬(nat≺nat) by auto
    then have (CoFinite nat){is hyperconnected} using Cofinite_nat_HConn[of
nat] by auto
    ultimately have nat{is in the spectrum of}IsHConnected unfolding antiProperty_def
by auto
    then have nat≲1 using HConn_spectrum by auto
    moreover
    have 1∈nat by auto
    then have 1≺nat using n_lesspoll_nat by auto
    ultimately have nat≺nat using lesspoll_trans1 by auto
    then have False by auto
  }
  then show ¬((CoFinite nat){is anti-}IsHConnected) by auto
next
  show (CoFinite nat){is T₁} using cocardinal_is_T1 InfCard_nat unfold-
ing Cofinite_def by auto
qed
```

The join-topology build from the cofinite topology on the natural numbers, and the excluded set topology on the natural numbers excluding {0,1}; is just the union of both.

```
lemma join_top_cofinite_excluded_set:
  shows (joinT {CoFinite nat,ExcludedSet(nat,{0,1})})=(CoFinite nat)∪
ExcludedSet(nat,{0,1})
proof-
  have coftop:(CoFinite nat){is a topology} unfolding Cofinite_def us-
ing CoCar_is_topology InfCard_nat by auto
  moreover
  have ExcludedSet(nat,{0,1}){is a topology} using excludedset_is_topology
by auto
  moreover
  have exuni:⋃ExcludedSet(nat,{0,1})=nat using union_excludedset by auto
  moreover
```

```
  have cofuni:⋃(CoFinite nat)=nat using union_cocardinal unfolding Cofinite_def
by auto
  ultimately have (joinT {CoFinite nat,ExcludedSet(nat,{0,1})}) = (THE
T. (CoFinite nat)∪ExcludedSet(nat,{0,1}) {is a subbase for} T)
    using joinT_def by auto
  moreover
  have ⋃(CoFinite nat)∈CoFinite nat using CoCar_is_topology[OF InfCard_nat]
unfolding Cofinite_def IsATopology_def
    by auto
  with cofuni have n:nat∈CoFinite nat by auto
  have Pa:(CoFinite nat)∪ExcludedSet(nat,{0,1}) {is a subbase for}{⋃A.
A∈Pow({⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0,1}))})}
    using Top_subbase(2) by auto
  have {⋃A. A∈Pow({⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0,1}))})}=(THE
T. (CoFinite nat)∪ExcludedSet(nat,{0,1}) {is a subbase for} T)
    using same_subbase_same_top[where B=(CoFinite nat)∪ExcludedSet(nat,{0,1}),
OF _ Pa] the_equality[where a={⋃A. A∈Pow({⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0
and P=λT. ((CoFinite nat)∪ExcludedSet(nat,{0,1})){is a subbase for}T,
      OF Pa] by auto
  ultimately have equal:(joinT {CoFinite nat,ExcludedSet(nat,{0,1})})
={⋃A. A∈Pow({⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0,1}))})}
    by auto
  {
    fix U assume U∈{⋃A. A∈Pow({⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0,1}))})}
    then obtain AU where U=⋃AU and base:AU∈Pow({⋂B. B∈FinPow((CoFinite
nat)∪ExcludedSet(nat,{0,1}))})
      by auto
    have (CoFinite nat)⊆Pow(⋃(CoFinite nat)) by auto
    moreover
    have ExcludedSet(nat,{0,1})⊆Pow(⋃ExcludedSet(nat,{0,1})) by auto
    moreover
    note cofuni exuni
    ultimately have sub:(CoFinite nat)∪ExcludedSet(nat,{0,1})⊆Pow(nat)
by auto
    from base have ∀S∈AU. S∈{⋂B. B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0,1}))}
by blast
    then have ∀S∈AU. ∃B∈FinPow((CoFinite nat)∪ExcludedSet(nat,{0,1})).
S=⋂B by blast
    then have eq:∀S∈AU. ∃B∈Pow((CoFinite nat)∪ExcludedSet(nat,{0,1})).
S=⋂B unfolding FinPow_def by blast
    {
      fix S assume S∈AU
      with eq obtain B where B∈Pow((CoFinite nat)∪ExcludedSet(nat,{0,1}))S=⋂B
by auto
      with sub have B∈Pow(Pow(nat)) by auto
      {
        fix x assume x∈⋂B
        then have ∀N∈B. x∈NB≠0 by auto
        with ‹B∈Pow(Pow(nat))› have x∈nat by blast
```

```
      }
      with ‹S=⋂B› have S∈Pow(nat) by auto
    }
    then have ∀S∈AU. S∈Pow(nat) by blast
    with ‹U=⋃AU› have U∈Pow(nat) by auto
    {
      assume 0∈U∨1∈U
      with ‹U=⋃AU› obtain S where S∈AU0∈S∨1∈S by auto
      with base obtain BS where S=⋂BS and bsbase:BS∈FinPow((CoFinite
nat)∪ExcludedSet(nat,{0,1})) by auto
      with ‹0∈S∨1∈S› have ∀M∈BS. 0∈M∨1∈M by auto
      then have ∀M∈BS. M∉ExcludedSet(nat,{0,1})-{nat} unfolding ExcludedPoint_def
ExcludedSet_def by auto
      moreover
      note bsbase n
      ultimately have BS∈FinPow(CoFinite nat) unfolding FinPow_def by
auto
      moreover
      from ‹0∈S∨1∈S› have S≠0 by auto
      with ‹S=⋂BS› have BS≠0 by auto
      moreover
      note coftop
      ultimately have ⋂BS∈CoFinite nat using topology0.fin_inter_open_open[OF
topology0_CoCardinal[OF InfCard_nat]]
        unfolding Cofinite_def by auto
      with ‹S=⋂BS› have S∈CoFinite nat by auto
      with ‹0∈S∨1∈S› have nat-S≺nat unfolding Cofinite_def CoCardinal_def
by auto
      moreover
      from ‹U=⋃AU›‹S∈AU› have S⊆U by auto
      then have nat-U⊆nat-S by auto
      then have nat-U≲nat-S using subset_imp_lepoll by auto
      ultimately
      have nat-U≺nat using lesspoll_trans1 by auto
      with ‹U∈Pow(nat)› have U∈CoFinite nat unfolding Cofinite_def CoCardinal_def
by auto
      with ‹U∈Pow(nat)› have U∈ (CoFinite nat)∪ ExcludedSet(nat,{0,1})
by auto
    }
    with ‹U∈Pow(nat)› have U∈(CoFinite nat)∪ ExcludedSet(nat,{0,1}) un-
folding ExcludedSet_def by blast
  }
  then have ({⋃A . A ∈ Pow({⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,{0,1}))})})
⊆ (CoFinite nat)∪ ExcludedSet(nat,{0,1})
    by blast
  moreover
  {
    fix U
    assume U∈(CoFinite nat)∪ ExcludedSet(nat,{0,1})
```

then have {U}∈FinPow((CoFinite nat) ∪ ExcludedSet(nat,{0,1})) **unfolding** FinPow_def **by** auto
then have {U}∈Pow({⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,{0,1}))}) **by** blast
moreover
have U=⋃{U} **by** auto
ultimately have U∈{⋃A . A ∈ Pow({⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,{0,1}))})} **by** blast
}
then have (CoFinite nat)∪ ExcludedSet(nat,{0,1})⊆{⋃A . A ∈ Pow({⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,{0,1}))})}
by auto
ultimately have (CoFinite nat)∪ ExcludedSet(nat,{0,1})={⋃A . A ∈ Pow({⋂B . B ∈ FinPow((CoFinite nat) ∪ ExcludedSet(nat,{0,1}))})}
by auto
with equal **show** thesis **by** auto
qed

The previous topology in not T_2, but is anti-hyperconnected.

theorem join_Cofinite_ExclPoint_not_T2:
shows
¬((joinT {CoFinite nat, ExcludedSet(nat,{0,1})}){is T_2}) **and**
(joinT {CoFinite nat,ExcludedSet(nat,{0,1})}){is anti-} IsHConnected
proof−
have (CoFinite nat) ⊆ (CoFinite nat) ∪ ExcludedSet(nat,{0,1}) **by** auto
have ⋃((CoFinite nat)∪ ExcludedSet(nat,{0,1}))=(⋃(CoFinite nat))∪ (⋃ExcludedSet(nat,{0,1}))
by auto
moreover
have ...=nat **unfolding** Cofinite_def **using** union_cocardinal union_excludedset **by** auto
ultimately have tot:⋃((CoFinite nat)∪ ExcludedSet(nat,{0,1}))=nat **by** auto
{
assume (joinT {CoFinite nat,ExcludedSet(nat,{0,1})}) {is T_2}
then have t2:((CoFinite nat)∪ ExcludedSet(nat,{0,1})){is T_2} **using** join_top_cofinite_excluded_set
by auto
with tot **have** ∃U∈((CoFinite nat)∪ ExcludedSet(nat,{0,1})). ∃V∈((CoFinite nat)∪ ExcludedSet(nat,{0,1})). 0∈U∧1∈V∧U∩V=0 **using** isT2_def **by** auto
then obtain U V **where** U ∈ (CoFinite nat) ∨ (0 ∉ U∧1∉U)V ∈ (CoFinite nat) ∨ (0 ∉ V∧1∉V)0∈U1∈VU∩V=0
unfolding ExcludedSet_def **by** auto
then have U∈(CoFinite nat)V∈(CoFinite nat) **by** auto
with ‹0∈U›‹1∈V› **have** U∩V≠0 **using** Cofinite_nat_HConn IsHConnected_def **by** auto
with ‹U∩V=0› **have** False **by** auto
}
then show ¬((joinT {CoFinite nat,ExcludedSet(nat,{0,1})}){is T_2}) **by**

```
auto
  {
    fix A assume AS:A∈Pow(⋃((CoFinite nat)∪ ExcludedSet(nat,{0,1})))(((CoFinite
nat)∪ ExcludedSet(nat,{0,1})){restricted to}A){is hyperconnected}
    with tot have A∈Pow(nat) by auto
    then have sub:A∩nat=A by auto
    have ((CoFinite nat)∪ ExcludedSet(nat,{0,1})){restricted to}A=((CoFinite
nat){restricted to}A)∪ (ExcludedSet(nat,{0,1}){restricted to}A)
      unfolding RestrictedTo_def by auto
    also from sub have ...=(CoFinite A)∪ExcludedSet(A,{0,1}) using subspace_excludedset[ofna
subspace_cocardinal[of natnatA] unfolding Cofinite_def
      by auto
    finally have ((CoFinite nat)∪ ExcludedSet(nat,{0,1})){restricted to}A=(CoFinite
A)∪ExcludedSet(A,{0,1}) by auto
    with AS(2) have eq:((CoFinite A)∪ExcludedSet(A,{0,1})){is hyperconnected}
by auto
    {
      assume {0,1}∩A=0
      then have (CoFinite A)∪ExcludedSet(A,{0,1})=Pow(A) using empty_excludedset[of
{0,1}A] unfolding Cofinite_def CoCardinal_def
        by auto
      with eq have Pow(A){is hyperconnected} by auto
      then have Pow(A){is connected} using HConn_imp_Conn by auto
      moreover
      have Pow(A){is anti-}IsConnected using discrete_tot_dis unfold-
ing IsTotDis_def by auto
      moreover
      have ⋃(Pow(A))∈Pow(⋃(Pow(A))) by auto
      moreover
      have Pow(A){restricted to}⋃(Pow(A))=Pow(A) unfolding RestrictedTo_def
by blast
      ultimately have (⋃(Pow(A))){is in the spectrum of}IsConnected un-
folding antiProperty_def
        by auto
      then have A{is in the spectrum of}IsConnected by auto
      then have A≲1 using conn_spectrum  by auto
      then have A{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
    }
    moreover
    {
      assume AS:{0,1}∩A≠0
      {
        assume A={0}∨A={1}
        then have A≈1 using singleton_eqpoll_1 by auto
        then have A≲1 using eqpoll_imp_lepoll by auto
        then have A{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
      }
```

```
      moreover
      {
        assume AS2:¬(A={0}∨A={1})
        {
          assume AS3:A⊆{0,1}
          with AS AS2 have A_def:A={0,1} by blast
          then have ExcludedSet(A,{0,1})=ExcludedSet(A,A) by auto
          moreover have ExcludedSet(A,A)={0,A} unfolding ExcludedSet_def
by blast
          ultimately have ExcludedSet(A,{0,1})={0,A} by auto
          moreover
          have 0∈(CoFinite A) using empty_open[of CoFinite A]
            CoCar_is_topology[OF InfCard_nat,of A] unfolding Cofinite_def
by auto
          moreover
          have ⋃(CoFinite A)=A using union_cocardinal unfolding Cofinite_def
by auto
          then have A∈(CoFinite A) using CoCar_is_topology[OF InfCard_nat,of
A] unfolding Cofinite_def
            IsATopology_def by auto
          ultimately have (CoFinite A)∪ExcludedSet(A,{0,1})=(CoFinite
A) by auto
          with eq have(CoFinite A){is hyperconnected} by auto
          with A_def have  hyp:(CoFinite {0,1}){is hyperconnected} by
auto
          have {0}≈1{1}≈1 using singleton_eqpoll_1 by auto
          moreover
          have 1≺nat using n_lesspoll_nat by auto
          ultimately have {0}≺nat{1}≺nat using eq_lesspoll_trans by auto
          moreover
          have {0,1}-{1}={0}{0,1}-{0}={1} by auto
          ultimately have {1}∈(CoFinite {0,1}){0}∈(CoFinite {0,1}) {1}∩{0}=0
unfolding Cofinite_def CoCardinal_def
            by auto
          with hyp have False unfolding IsHConnected_def by auto
        }
        then obtain t where t∈A t≠0 t≠1 by auto
        then have {t}∈ExcludedSet(A,{0,1}) unfolding ExcludedSet_def
by auto
        moreover
        {
          have {t}≈1 using singleton_eqpoll_1 by auto
          moreover
          have 1≺nat using n_lesspoll_nat by auto
          ultimately have {t}≺nat using eq_lesspoll_trans by auto
          moreover
          with ‹t∈A› have A-(A-{t})={t} by auto
          ultimately have A-{t}∈(CoFinite A) unfolding Cofinite_def CoCardinal_def
            by auto
```

```
        }
        ultimately have {t}∈((CoFinite A)∪ExcludedSet(A,{0,1}))A-{t}∈((CoFinite
A)∪ExcludedSet(A,{0,1}))
          {t}∩(A-{t})=0 by auto
        with eq have A-{t}=0 unfolding IsHConnected_def by auto
        with ‹t∈A› have A={t} by auto
        then have A≈1 using singleton_eqpoll_1 by auto
        then have A≲1 using eqpoll_imp_lepoll by auto
        then have A{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
      }
      ultimately have A{is in the spectrum of}IsHConnected by auto
    }
    ultimately have A{is in the spectrum of}IsHConnected by auto
  }
  then have ((CoFinite nat)∪ExcludedSet(nat,{0,1})){is anti-}IsHConnected
unfolding antiProperty_def
    by auto
  then show (joinT {CoFinite nat, ExcludedSet(nat,{0,1})}){is anti-}IsHConnected
using join_top_cofinite_excluded_set
    by auto
qed
```

Let's show that anti-hyperconnected is in fact T_1 and sober. The trick of the proof lies in the fact that if a subset is hyperconnected, its closure is so too (the closure of a point is then always hyperconnected because singletons are in the spectrum); since the closure is closed, we can apply the sober property on it.

```
theorem (in topology0) T1_sober_imp_anti_HConn:
  assumes T{is T₁} and T{is sober}
  shows T{is anti-}IsHConnected
proof-
  {
    fix A assume AS:A∈Pow(⋃T)(T{restricted to}A){is hyperconnected}
    {
      assume A=0
      then have A≲1 using empty_lepollI by auto
      then have A{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
    }
    moreover
    {
      assume A≠0
      then obtain x where x∈A by blast
      {
        assume ¬((T{restricted to}Closure(A,T)){is hyperconnected})
        then obtain U V where UV_def:U∈(T{restricted to}Closure(A,T))V∈(T{restricted
to}Closure(A,T))
          U∩V=0U≠0V≠0 using IsHConnected_def by auto
```

```
        then obtain UCA VCA where UCA∈TVCA∈TU=UCA∩Closure(A,T)V=VCA∩Closure(A,T)
          unfolding RestrictedTo_def by auto
        from ‹A∈Pow(⋃T)› have A⊆Closure(A,T) using cl_contains_set by
auto
        then have UCA∩A⊆UCA∩Closure(A,T)VCA∩A⊆VCA∩Closure(A,T) by auto
        with ‹U=UCA∩Closure(A,T)›‹V=VCA∩Closure(A,T)›‹U∩V=0› have (UCA∩A)∩(VCA∩A)=0
by auto
        moreover
        from ‹UCA∈T›‹VCA∈T› have UCA∩A∈(T{restricted to}A)VCA∩A∈(T{restricted
to}A)
          unfolding RestrictedTo_def by auto
        moreover
        note AS(2)
        ultimately have UCA∩A=0∨VCA∩A=0 using IsHConnected_def by auto
        with ‹A⊆Closure(A,T)› have A⊆Closure(A,T)-UCA∨A⊆Closure(A,T)-VCA
by auto
        moreover
        {
          have Closure(A,T)-UCA=Closure(A,T)∩(⋃T-UCA)Closure(A,T)-VCA=Closure(A,T)∩(⋃T-VC
            using Top_3_L11(1) AS(1) by auto
          moreover
          with ‹UCA∈T›‹VCA∈T› have (⋃T-UCA){is closed in}T(⋃T-VCA){is
closed in}TClosure(A,T){is closed in}T
            using Top_3_L9 cl_is_closed AS(1) by auto
          ultimately have (Closure(A,T)-UCA){is closed in}T(Closure(A,T)-VCA){is
closed in}T
            using Top_3_L5(1) by auto
        }
        ultimately
        have Closure(A,T)⊆Closure(A,T)-UCA∨Closure(A,T)⊆Closure(A,T)-VCA
using Top_3_L13
          by auto
        then have UCA∩Closure(A,T)=0∨VCA∩Closure(A,T)=0 by auto
        with ‹U=UCA∩Closure(A,T)›‹V=VCA∩Closure(A,T)› have U=0∨V=0 by
auto
        with ‹U≠0›‹V≠0› have False by auto
      }
      then have (T{restricted to}Closure(A,T)){is hyperconnected} by
auto
      moreover
      have Closure(A,T){is closed in}T using cl_is_closed AS(1) by auto
      moreover
      from ‹x∈A› have Closure(A,T)≠0 using cl_contains_set AS(1) by
auto
      moreover
      from AS(1) have Closure(A,T)⊆⋃T using Top_3_L11(1) by auto
      ultimately have Closure(A,T)∈Pow(⋃T)-{0}(T {restricted to} Closure(A,
T)){is hyperconnected} Closure(A, T) {is closed in} T
        by auto
```

```
      moreover note assms(2)
      ultimately have ∃x∈⋃T. (Closure(A,T)=Closure({x},T)∧ (∀y∈⋃T.
Closure(A,T) = Closure({y}, T) ⟶ y = x)) unfolding IsSober_def
        by auto
      then obtain y where y∈⋃TClosure(A,T)=Closure({y},T) by auto
      moreover
      {
        fix z assume z∈(⋃T)-{y}
        with assms(1) ‹y∈⋃T› obtain U where U∈T z∈U y∉U using isT1_def
by blast
        then have U∈T z∈U U⊆(⋃T)-{y} by auto
        then have ∃U∈T. z∈U ∧ U⊆(⋃T)-{y} by auto
      }
      then have ∀z∈(⋃T)-{y}. ∃U∈T. z∈U ∧ U⊆(⋃T)-{y} by auto
      then have ⋃T-{y}∈T using open_neigh_open by auto
      with ‹y∈⋃T› have {y} {is closed in}T using IsClosed_def by auto
      with ‹y∈⋃T› have Closure({y},T)={y} using Top_3_L8 by auto
      with ‹Closure(A,T)=Closure({y},T)› have Closure(A,T)={y} by auto
      with AS(1) have A⊆{y} using cl_contains_set[of A] by auto
      with ‹A≠0› have A={y} by auto
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
      then have A{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
    }
    ultimately have A{is in the spectrum of}IsHConnected by blast
  }
  then show thesis using antiProperty_def by auto
qed

theorem (in topology0) anti_HConn_iff_T1_sober:
  shows (T{is anti-}IsHConnected) ⟷ (T{is sober}∧T{is T₁})
  using T1_sober_imp_anti_HConn anti_HConn_imp_T1 anti_HConn_imp_sober
by auto
```

A space is ultraconnected iff every two non-empty closed sets meet.

```
definition IsUConnected (_{is ultraconnected}80)
  where T{is ultraconnected}≡ ∀A B. A{is closed in}T∧B{is closed in}T∧A∩B=0
⟶ A=0∨B=0
```

Every ultraconnected space is trivially normal.

```
lemma (in topology0)UConn_imp_normal:
  assumes T{is ultraconnected}
  shows T{is normal}
proof-
  {
    fix A B
    assume AS:A{is closed in}T B{is closed in}TA∩B=0
    with assms have A=0∨B=0 using IsUConnected_def by auto
```

```
    with AS(1,2) have (A⊆0∧B⊆⋃T)∨(A⊆⋃T∧B⊆0) unfolding IsClosed_def
by auto
    moreover
    have 0∈T using empty_open topSpaceAssum by auto
    moreover
    have ⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto
    ultimately have ∃U∈T. ∃V∈T. A⊆U∧B⊆V∧U∩V=0 by auto
  }
  then show thesis unfolding IsNormal_def by auto
qed
```

Every ultraconnected space is connected.

```
lemma UConn_imp_Conn:
  assumes T{is ultraconnected}
  shows T{is connected}
proof-
  {
    fix U V
    assume U∈TU{is closed in}T
    then have ⋃T-(⋃T-U)=U by auto
    with ‹U∈T› have (⋃T-U){is closed in}T unfolding IsClosed_def by
auto
    with ‹U{is closed in}T› assms have U=0∨⋃T-U=0 unfolding IsUConnected_def
by auto
    with ‹U∈T› have U=0∨U=⋃T by auto
  }
  then show thesis unfolding IsConnected_def by auto
qed

lemma UConn_spectrum:
  shows (A{is in the spectrum of}IsUConnected) ⟷ A≲1
proof
  assume A_spec:(A{is in the spectrum of}IsUConnected)
  {
    assume A=0
    then have A≲1 using empty_lepollI by auto
  }
  moreover
  {
    assume A≠0
    from A_spec have ∀T. (T{is a topology}∧⋃T≈A) ⟶ (T{is ultraconnected})
unfolding Spec_def by auto
    moreover
    have Pow(A){is a topology} using Pow_is_top by auto
    moreover
    have ⋃Pow(A)=A by auto
    then have ⋃Pow(A)≈A by auto
    ultimately have ult:Pow(A){is ultraconnected} by auto
    moreover
```

```
    from ‹A≠0› obtain b where b∈A by auto
    then have {b}{is closed in}Pow(A) unfolding IsClosed_def by auto
    {
      fix c
      assume c∈Ac≠b
      then have {c}{is closed in}Pow(A){c}∩{b}=0 unfolding IsClosed_def
by auto
      with ult ‹{b}{is closed in}Pow(A)› have False using IsUConnected_def
by auto
    }
    with ‹b∈A› have A={b} by auto
    then have A≈1 using singleton_eqpoll_1 by auto
    then have A≲1 using eqpoll_imp_lepoll by auto
  }
  ultimately show A≲1 by auto
next
  assume A≲1
  {
    fix T
    assume T{is a topology}⋃T≈A
    {
      assume ⋃T=0
      with ‹T{is a topology}› have T={0} using empty_open by auto
      then have T{is ultraconnected} unfolding IsUConnected_def IsClosed_def
by auto
    }
    moreover
    {
      assume ⋃T≠0
      moreover
      from ‹A≲1›‹⋃T≈A› have ⋃T≲1 using eq_lepoll_trans by auto
      ultimately
      obtain E where eq:⋃T={E} using lepoll_1_is_sing by blast
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow({E}) by auto
      then have T⊆{0,{E}} by blast
      with ‹T{is a topology}› have {0}⊆T T⊆{0,{E}} using empty_open by
auto
      then have T{is ultraconnected} unfolding IsUConnected_def IsClosed_def
by (simp only: eq, safe, force)
    }
    ultimately have T{is ultraconnected} by auto
  }
  then show A{is in the spectrum of}IsUConnected unfolding Spec_def by
auto
qed
```

This time, anti-ultraconnected is an old property.

```
theorem (in topology0) anti_UConn:
  shows (T{is anti-}IsUConnected) ⟷ T{is T_1}
proof
  assume T{is T_1}
  {
    fix TT
    {
      assume TT{is a topology}TT{is T_1}TT{is ultraconnected}
      {
        assume ⋃TT=0
        then have ⋃TT≲1 using empty_lepollI by auto
        then have ((⋃TT){is in the spectrum of}IsUConnected) using UConn_spectrum
by auto
      }
      moreover
      {
        assume ⋃TT≠0
        then obtain t where t∈⋃TT by blast
        {
          fix x
          assume p:x∈⋃TT
          {
            fix y assume y∈(⋃TT)-{x}
            with ‹TT{is T_1}› p obtain U where U∈TT y∈U x∉U using isT1_def
by blast
            then have U∈TT y∈U U⊆(⋃TT)-{x} by auto
            then have ∃U∈TT. y∈U ∧ U⊆(⋃TT)-{x} by auto
          }
          then have ∀y∈(⋃TT)-{x}. ∃U∈TT. y∈U ∧ U⊆(⋃TT)-{x} by auto
          with ‹TT{is a topology}› have ⋃TT-{x}∈TT using topology0.open_neigh_open
unfolding topology0_def by auto
          with p have {x} {is closed in}TT using IsClosed_def by auto
        }
        then have reg:∀x∈⋃TT. {x}{is closed in}TT by auto
        with ‹t∈⋃TT› have t_cl:{t}{is closed in}TT by auto
        {
          fix y
          assume y∈⋃TT
          with reg have {y}{is closed in}TT by auto
          with ‹TT{is ultraconnected}› t_cl have y=t unfolding IsUConnected_def
by auto
        }
        with ‹t∈⋃TT› have ⋃TT={t} by blast
        then have ⋃TT≈1 using singleton_eqpoll_1 by auto
        then have ⋃TT≲1 using eqpoll_imp_lepoll by auto
        then have (⋃TT){is in the spectrum of}IsUConnected using UConn_spectrum
by auto
      }
      ultimately have (⋃TT){is in the spectrum of}IsUConnected by blast
```

```
    }
    then have (TT{is a topology}∧TT{is T₁}∧(TT{is ultraconnected}))⟶
((⋃TT){is in the spectrum of}IsUConnected)
      by auto
  }
  then have ∀TT. (TT{is a topology}∧TT{is T₁}∧(TT{is ultraconnected}))⟶
((⋃TT){is in the spectrum of}IsUConnected)
    by auto
  moreover
  note here_T1
  ultimately have ∀T. T{is a topology} ⟶ ((T{is T₁})⟶(T{is anti-}IsUConnected))
using Q_P_imp_Spec[where Q=isT1 and P=IsUConnected]
    by auto
  with topSpaceAssum have (T{is T₁})⟶(T{is anti-}IsUConnected) by auto
  with ‹T{is T₁}› show T{is anti-}IsUConnected by auto
next
  assume ASS:T{is anti-}IsUConnected
  {
    fix x y
    assume x∈⋃Ty∈⋃Tx≠y
    then have tot:⋃(T{restricted to}{x,y})={x,y} unfolding RestrictedTo_def
by auto
    {
      assume AS:∀U∈T. x∈U⟶y∈U
      {
        assume {y}{is closed in}(T{restricted to}{x,y})
        moreover
        from ‹x≠y› have {x,y}-{y}={x} by auto
        ultimately have {x}∈(T{restricted to}{x,y}) unfolding IsClosed_def
by (simp only:tot)
        then obtain U where U∈T{x}={x,y}∩U unfolding RestrictedTo_def
by auto
        moreover
        with ‹x≠y› have y∉{x} y∈{x,y} by (blast+)
        with ‹{x}={x,y}∩U› have y∉U by auto
        moreover have x∈{x} by auto
        with ‹{x}={x,y}∩U› have x∈U by auto
        ultimately have x∈Uy∉UU∈T by auto
        with AS have False by auto
      }
      then have y_no_cl:¬({y}{is closed in}(T{restricted to}{x,y})) by
auto
      {
        fix A B
        assume cl:A{is closed in}(T{restricted to}{x,y})B{is closed in}(T{restricted
to}{x,y})A∩B=0
        with tot have A⊆{x,y}B⊆{x,y}A∩B=0 unfolding IsClosed_def by
auto
        then have x∈A⟶x∉By∈A⟶y∉BA⊆{x,y}B⊆{x,y} by auto
```

```
        {
          assume x∈A
          with ‹x∈A⟶x∉B›‹B⊆{x,y}› have B⊆{y} by auto
          then have B=0∨B={y} by auto
          with y_no_cl cl(2) have B=0 by auto
        }
        moreover
        {
          assume x∉A
          with ‹A⊆{x,y}› have A⊆{y} by auto
          then have A=0∨A={y} by auto
          with y_no_cl cl(1) have A=0 by auto
        }
        ultimately have A=0∨B=0 by auto
      }
      then have (T{restricted to}{x,y}){is ultraconnected} unfolding IsUConnected_def
by auto
      with ASS ‹x∈⋃T›‹y∈⋃T› have {x,y}{is in the spectrum of}IsUConnected
unfolding antiProperty_def
        by auto
      then have {x,y}≲1 using UConn_spectrum by auto
      moreover have x∈{x,y} by auto
      ultimately have {x}={x,y} using lepoll_1_is_sing[of {x,y}x] by auto
      moreover
      have y∈{x,y} by auto
      ultimately have y∈{x} by auto
      then have y=x by auto
      then have False using ‹x≠y› by auto
    }
    then have ∃U∈T. x∈U∧y∉U by auto
  }
  then show T{is T₁} unfolding isT1_def by auto
qed
```

Is is natural that separation axioms and connection axioms are anti-properties of each other; as the concepts of connectedness and separation are opposite.

To end this section, let's try to charaterize anti-sober spaces.

```
lemma sober_spectrum:
  shows (A{is in the spectrum of}IsSober) ⟷ A≲1
proof
  assume AS:A{is in the spectrum of}IsSober
  {
    assume A=0
    then have A≲1 using empty_lepollI by auto
  }
  moreover
  {
    assume A≠0
```

```
    note AS
    moreover
    have top:{0,A}{is a topology} unfolding IsATopology_def by auto
    moreover
    have ⋃{0,A}=A by auto
    then have ⋃{0,A}≈A by auto
    ultimately have {0,A}{is sober} using Spec_def by auto
    moreover
    have {0,A}{is hyperconnected} using Indiscrete_HConn by auto
    moreover
    have {0,A}{restricted to}A={0,A} unfolding RestrictedTo_def by auto
    moreover
    have A{is closed in}{0,A} unfolding IsClosed_def by auto
    moreover
    note ‹A≠0›
    ultimately have ∃x∈A. A=Closure({x},{0,A})∧ (∀y∈⋃{0, A}. A = Closure({y},
{0, A}) ⟶ y = x) unfolding IsSober_def by auto
    then obtain x where x∈A A=Closure({x},{0,A}) and reg:∀y∈A. A = Closure({y},
{0, A}) ⟶ y = x by auto
    {
      fix y assume y∈A
      with top have Closure({y},{0,A}){is closed in}{0,A} using topology0.cl_is_closed
        topology0_def by auto
      moreover
      from ‹y∈A› top have y∈Closure({y},{0,A}) using topology0.cl_contains_set
        topology0_def by auto
      ultimately have A-Closure({y},{0,A})∈{0,A}Closure({y},{0,A})∩A≠0
unfolding IsClosed_def
        by auto
      then have A-Closure({y},{0,A})=A∨A-Closure({y},{0,A})=0
        by auto
      moreover
      from ‹y∈A›‹y∈Closure({y},{0,A})› have y∈Ay∉A-Closure({y},{0,A})
by auto
      ultimately have A-Closure({y},{0,A})=0 by (cases A-Closure({y},{0,A})=A,
simp, auto)
      moreover
      from ‹y∈A› top have Closure({y},{0,A})⊆A using topology0_def topology0.Top_3_L11(1)
by blast
      then have A-(A-Closure({y},{0,A}))=Closure({y},{0,A}) by auto
      ultimately have A=Closure({y},{0,A}) by auto
    }
    with reg have ∀y∈A. x=y by auto
    with ‹x∈A› have A={x} by blast
    then have A≈1 using singleton_eqpoll_1 by auto
    then have A≲1 using eqpoll_imp_lepoll by auto
  }
  ultimately show A≲1 by auto
next
```

```
  assume A≲1
  {
    fix T assume T{is a topology}⋃T≈A
    {
      assume ⋃T=0
      then have T{is sober} unfolding IsSober_def by auto
    }
    moreover
    {
      assume ⋃T≠0
      then obtain x where x∈⋃T by blast
      moreover
      from ‹⋃T≈A› ‹A≲1› have ⋃T≲1 using eq_lepoll_trans by auto
      ultimately have ⋃T={x} using lepoll_1_is_sing by auto
      moreover
      have T⊆Pow(⋃T) by auto
      ultimately have T⊆Pow({x}) by auto
      then have T⊆{0,{x}} by blast
      moreover
      from ‹T{is a topology}› have 0∈T using empty_open by auto
      moreover
      from ‹T{is a topology}› have ⋃T∈T unfolding IsATopology_def by
auto
      with ‹⋃T={x}› have {x}∈T by auto
      ultimately have T_def:T={0,{x}} by auto
      then have dd:Pow(⋃T)-{0}={{x}} by auto
      {
        fix B assume B∈Pow(⋃T)-{0}
        with dd have B_def:B={x} by auto
        from ‹T{is a topology}› have (⋃T){is closed in}T using topology0_def
topology0.Top_3_L1
          by auto
        with ‹⋃T={x}› ‹T{is a topology}› have Closure({x},T)={x} us-
ing topology0.Top_3_L8
          unfolding topology0_def by auto
        with B_def have B=Closure({x},T) by auto
        moreover
        {
          fix y assume y∈⋃T
          with ‹⋃T={x}› have y=x by auto
          }
        then have (∀y∈⋃T. B = Closure({y}, T) ⟶ y = x) by auto
        moreover note ‹x∈⋃T›
        ultimately have (∃x∈⋃T. B = Closure({x}, T) ∧ (∀y∈⋃T. B = Closure({y},
T) ⟶ y = x))
          by auto
      }
      then have T{is sober} unfolding IsSober_def by auto
    }
```

```
    ultimately have T{is sober} by blast
  }
  then show A {is in the spectrum of} IsSober unfolding Spec_def by auto
qed

theorem (in topology0)anti_sober:
  shows (T{is anti-}IsSober) ⟷ T={0,⋃T}
proof
  assume T={0,⋃T}
  {
    fix A assume A∈Pow(⋃T)(T{restricted to}A){is sober}
    {
      assume A=0
      then have A≲1 using empty_lepollI by auto
      then have A{is in the spectrum of}IsSober using sober_spectrum
by auto
    }
    moreover
    {
      assume A≠0
      have ⋃T∈{0,⋃T}0∈{0,⋃T} by auto
      with ‹T={0,⋃T}› have (⋃T)∈T 0∈T by auto
      with ‹A∈Pow(⋃T)› have {0,A}⊆(T{restricted to}A) unfolding RestrictedTo_def
by auto
      moreover
      have ∀B∈{0,⋃T}. B=0∨B=⋃T by auto
      with ‹T={0,⋃T}› have ∀B∈T. B=0∨B=⋃T by auto
      with ‹A∈Pow(⋃T)› have T{restricted to}A⊆{0,A} unfolding RestrictedTo_def
by auto
      ultimately have top_def:T{restricted to}A={0,A} by auto
      moreover
      have A{is closed in}{0,A} unfolding IsClosed_def by auto
      moreover
      have {0,A}{is hyperconnected} using Indiscrete_HConn by auto
      moreover
      from ‹A∈Pow(⋃T)› have (T{restricted to}A){restricted to}A=T{restricted
to}A using subspace_of_subspace[of AAT]
        by auto
      moreover
      note ‹A≠0› ‹A∈Pow(⋃T)›
      ultimately have A∈Pow(⋃(T{restricted to}A))-{0}A{is closed in}(T{restricted
to}A)((T{restricted to}A){restricted to}A){is hyperconnected}
        by auto
      with ‹(T{restricted to}A){is sober}› have ∃x∈⋃(T{restricted to}A).
A=Closure({x},T{restricted to}A)∧(∀y∈⋃(T{restricted to}A). A=Closure({y},T{restricted
to}A) ⟶ y=x)
        unfolding IsSober_def by auto
      with top_def have ∃x∈A. A=Closure({x},{0,A})∧(∀y∈A. A=Closure({y},{0,A})
⟶ y=x) by auto
```

```
      then obtain x where x∈AA=Closure({x},{0,A})and reg:∀y∈A. A=Closure({y},{0,A})
⟶ y=x by auto
      {
        fix y assume y∈A
        from ‹A≠0› have top:{0,A}{is a topology} using indiscrete_ptopology[of
A] indiscrete_partition[of A] Ptopology_is_a_topology(1)[of {A}A]
          by auto
        with ‹y∈A› have Closure({y},{0,A}){is closed in}{0,A} using topology0.cl_is_closed
          topology0_def by auto
        moreover
        from ‹y∈A› top have y∈Closure({y},{0,A}) using topology0.cl_contains_set
          topology0_def by auto
        ultimately have A-Closure({y},{0,A})∈{0,A}Closure({y},{0,A})∩A≠0
unfolding IsClosed_def
          by auto
        then have A-Closure({y},{0,A})=A∨A-Closure({y},{0,A})=0
          by auto
        moreover
        from ‹y∈A›‹y∈Closure({y},{0,A})› have y∈Ay∉A-Closure({y},{0,A})
by auto
        ultimately have A-Closure({y},{0,A})=0 by (cases A-Closure({y},{0,A})=A,
simp, auto)
        moreover
        from ‹y∈A› top have Closure({y},{0,A})⊆A using topology0_def
topology0.Top_3_L11(1) by blast
        then have A-(A-Closure({y},{0,A}))=Closure({y},{0,A}) by auto
        ultimately have A=Closure({y},{0,A}) by auto
      }
      with reg ‹x∈A› have A={x} by blast
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
      then have A{is in the spectrum of}IsSober using sober_spectrum
by auto
    }
    ultimately have A{is in the spectrum of}IsSober by auto
  }
  then show T{is anti-}IsSober using antiProperty_def by auto
next
  assume T{is anti-}IsSober
  {
    fix A
    assume A∈TA≠0A≠⋃T
    then obtain x y where x∈Ay∈⋃T-A x≠yby blast
    then have {x}={x,y}∩A by auto
    with ‹A∈T› have {x}∈T{restricted to}{x,y} unfolding RestrictedTo_def
by auto
    {
      assume {y}∈T{restricted to}{x,y}
      from ‹y∈⋃T-A› ‹x∈A›‹A∈T› have ⋃(T{restricted to}{x,y})={x,y}
```

```
unfolding RestrictedTo_def
        by auto
      with ‹x≠y›‹{y}∈T{restricted to}{x,y}›‹{x}∈T{restricted to}{x,y}›
have (T{restricted to}{x,y}){is T2}
        unfolding isT2_def by auto
      then have (T{restricted to}{x,y}){is sober} using topology0.T2_imp_anti_HConn[of
T{restricted to}{x,y}]
        Top_1_L4 topology0_def topology0.anti_HConn_iff_T1_sober[of T{restricted
to}{x,y}] by auto
    }
    moreover
    {
      assume {y}∉T{restricted to}{x,y}
      moreover
      from ‹y∈⋃T-A› ‹x∈A›‹A∈T› have T{restricted to}{x,y}⊆Pow({x,y})
unfolding RestrictedTo_def by auto
      then have T{restricted to}{x,y}⊆{0,{x},{y},{x,y}} by blast
      moreover
      note ‹{x}∈T{restricted to}{x,y}› empty_open[OF Top_1_L4[of {x,y}]]
      moreover
      from ‹y∈⋃T-A› ‹x∈A›‹A∈T› have tot:⋃(T{restricted to}{x,y})={x,y}
unfolding RestrictedTo_def
        by auto
      from Top_1_L4[of {x,y}] have ⋃(T{restricted to}{x,y})∈T{restricted
to}{x,y} unfolding IsATopology_def
        by auto
      with tot have {x,y}∈T{restricted to}{x,y} by auto
      ultimately have top_d_def:T{restricted to}{x,y}={0,{x},{x,y}} by
auto
      {
        fix B assume B∈Pow({x,y})-{0}B{is closed in}(T{restricted to}{x,y})
        with top_d_def have (⋃(T{restricted to}{x,y}))-B∈{0,{x},{x,y}}
unfolding IsClosed_def by simp
        moreover have B∈{{x},{y},{x,y}} using ‹B∈Pow({x,y})-{0}› by
blast
        moreover note tot
        ultimately have {x,y}-B∈{0,{x},{x,y}} by auto
        have xin:x∈Closure({x},T{restricted to}{x,y}) using topology0.cl_contains_set[of
T{restricted to}{x,y}{x}]
          Top_1_L4[of {x,y}] unfolding topology0_def[of (T {restricted
to} {x, y})] using tot by auto
        {
          assume {x}{is closed in}(T{restricted to}{x,y})
          then have {x,y}-{x}∈(T{restricted to}{x,y}) unfolding IsClosed_def
using tot
            by auto
          moreover
          from ‹x≠y› have {x,y}-{x}={y} by auto
          ultimately have {y}∈(T{restricted to}{x,y}) by auto
```

```
      then have False using ‹{y}∉(T{restricted to}{x,y})› by auto
    }
    then have ¬({x}{is closed in}(T{restricted to}{x,y})) by auto
    moreover
    from tot have (Closure({x},T{restricted to}{x,y})){is closed
in}(T{restricted to}{x,y})
      using topology0.cl_is_closed unfolding topology0_def using Top_1_L4[of
{x,y}]
      tot by auto
    ultimately have ¬(Closure({x},T{restricted to}{x,y})={x}) by
auto
    moreover note xin topology0.Top_3_L11(1)[of T{restricted to}{x,y}{x}]
tot
    ultimately have cl_x:Closure({x},T{restricted to}{x,y})={x,y}
unfolding topology0_def
      using Top_1_L4[of {x,y}] by auto
    have {y}{is closed in}(T{restricted to}{x,y}) unfolding IsClosed_def
using tot
      top_d_def ‹x≠y› by auto
    then have cl_y:Closure({y},T{restricted to}{x,y})={y} using topology0.Top_3_L8[of
T{restricted to}{x,y}]
      unfolding topology0_def using Top_1_L4[of {x,y}] tot by auto
    {
      assume {x,y}-B=0
      with ‹B∈Pow({x,y})-{0}› have B:{x,y}=B by auto
      {
        fix m
        assume dis:m∈{x,y} and B_def:B=Closure({m},T{restricted
to}{x,y})
        {
          assume m=y
          with B_def have B=Closure({y},T{restricted to}{x,y}) by
auto
          with cl_y have B={y} by auto
          with B have {x,y}={y} by auto
          moreover have x∈{x,y} by auto
          ultimately
          have x∈{y} by auto
          with ‹x≠y› have False by auto
        }
        with dis have m=x by auto
      }
      then have (∀m∈{x,y}. B=Closure({m},T{restricted to}{x,y})⟶m=x
) by auto
      moreover
      have B=Closure({x},T{restricted to}{x,y}) using cl_x B by auto
      ultimately have ∃t∈{x,y}. B=Closure({t},T{restricted to}{x,y})
∧ (∀m∈{x,y}. B=Closure({m},T{restricted to}{x,y})⟶m=t )
        by auto
```

```
        }
        moreover
        {
          assume {x,y}-B≠0
          with ‹{x,y}-B∈{0,{x},{x,y}}› have or:{x,y}-B={x}∨{x,y}-B={x,y}
by auto
          {
            assume {x,y}-B={x}
            then have x∈{x,y}-B by auto
            with ‹B∈{{x},{y},{x,y}}› ‹x≠y› have B:B={y} by blast
            {
              fix m
              assume dis:m∈{x,y} and B_def:B=Closure({m},T{restricted
to}{x,y})
              {
                assume m=x
                with B_def have B=Closure({x},T{restricted to}{x,y})
by auto
                with cl_x have B={x,y} by auto
                with B have {x,y}={y} by auto
                moreover have x∈{x,y} by auto
                ultimately
                have x∈{y} by auto
                with ‹x≠y› have False by auto
              }
              with dis have m=y by auto
            }
            moreover
            have B=Closure({y},T{restricted to}{x,y}) using cl_y B by
auto
            ultimately have ∃t∈{x,y}. B=Closure({t},T{restricted to}{x,y})
∧ (∀m∈{x,y}. B=Closure({m},T{restricted to}{x,y})⟶m=t )
              by auto
          }
          moreover
          {
            assume {x,y}-B≠{x}
            with or have {x,y}-B={x,y} by auto
            then have x∈{x,y}-By∈{x,y}-B by auto
            with ‹B∈{{x},{y},{x,y}}› ‹x≠y› have False by auto
          }
          ultimately have ∃t∈{x,y}. B=Closure({t},T{restricted to}{x,y})
∧ (∀m∈{x,y}. B=Closure({m},T{restricted to}{x,y})⟶m=t )
            by auto
        }
        ultimately have ∃t∈{x,y}. B=Closure({t},T{restricted to}{x,y})
∧ (∀m∈{x,y}. B=Closure({m},T{restricted to}{x,y})⟶m=t )
          by auto
      }
```

```
      then have (T{restricted to}{x,y}){is sober} unfolding IsSober_def
using tot by auto
    }
    ultimately have (T{restricted to}{x,y}){is sober} by auto
    with ‹T{is anti-}IsSober› have {x,y}{is in the spectrum of}IsSober
unfolding antiProperty_def
      using ‹x∈A›‹A∈T›‹y∈⋃T-A› by auto
    then have {x,y}≲1 using sober_spectrum by auto
    moreover
    have x∈{x,y} by auto
    ultimately have {x,y}={x} using lepoll_1_is_sing[of {x,y}x] by auto
    moreover have y∈{x,y} by auto
    ultimately have y∈{x} by auto
    then have False using ‹x≠y› by auto
  }
  then have T⊆{0,⋃T} by auto
  with empty_open[OF topSpaceAssum] topSpaceAssum show T={0,⋃T} un-
folding IsATopology_def
    by auto
qed


end
```

84 Topology 8

```
theory Topology_ZF_8 imports Topology_ZF_6 EquivClass1
begin
```

Suppose T is a topology, r is an equivalence relation on $X = \bigcup T$ and $P_r : X \to X/r$ maps an element of X to its equivalence class $r\{x\}$. Then we can define a topology (on X/r) by taking the collection of those subsets V of X/r for which the inverse image by the projection P_r is in T. This is the weakest topology on X/r such that P_r is continuous. In this theory we consider a seemingly more general situation where we start with a topology T on $X = \bigcup T$ and a surjection $f : X \to Y$ and define a topology on Y by taking those subsets V of Y for which the inverse image by the mapping f is in T. Turns out that this construction is in a way equivalent to the previous one as the topology defined this way is homeomorphic to the topology defined by the equivalence relation r_f on X that relates two elements of X if f has the same value on them.

84.1 Definition of quotient topology

In this section we define the quotient topology generated by a topology T and a surjection $f : \bigcup T \to Y$, and show its basic properties.

For a topological space $X = \bigcup T$ and a surjection $f : X \to Y$ we define `{quotient topology in} Y {by} f` as the collection of subsets of Y whose inverse images by f are open.

```
definition (in topology0)
  QuotientTop ({quotient topology in}_{by}_ 80)
  where f∈surj(⋃T,Y) ⟹{quotient topology in} Y {by} f ≡
    {U∈Pow(Y). f-U∈T}
```

Outside of the `topology0` context we will indicate also the generating topology and write `{quotient topology in} Y {by} f {from} X`.

```
abbreviation QuotientTopTop ({quotient topology in}_{by}_{from}_)
  where {quotient topology in} Y {by} f {from} T ≡ topology0.QuotientTop(T,Y,f)
```

The quotient topology is indeed a topology.

```
theorem (in topology0) quotientTop_is_top:
  assumes f∈surj(⋃T,Y)
  shows ({quotient topology in} Y {by} f) {is a topology}
proof-
  have ({quotient topology in} Y {by} f)={U ∈ Pow(Y) . f-(U) ∈ T} us-
ing QuotientTop_def assms
    by auto
  moreover
  {
    fix M x B assume M: M ⊆ {U ∈ Pow(Y) . f-(U) ∈ T}
    then have ⋃M⊆Y by blast
    moreover have A1: f-(⋃M) = (⋃y∈(⋃M). f-{y}) using vimage_eq_UN
by blast
    moreover
    {
      fix A assume A∈M
      with M have A∈Pow(Y) f-(A) ∈ T by auto
      have f-(A) = (⋃y∈A. f-{y}) using vimage_eq_UN by blast
    }
    hence (⋃A∈M. f-(A)) = (⋃y∈⋃M. f-{y}) by auto
    with A1 have A2: f-(⋃M)=⋃{f- A. A∈M} by auto
    moreover
    {
      fix A assume A∈M
      with M have f-(A)∈T by auto
    }
    hence {f-(A). A∈M}⊆T by auto
    then have (⋃{f-(A). A∈M})∈T
      using topSpaceAssum unfolding IsATopology_def by auto
    with A2 have (f-(⋃M))∈T by auto
    ultimately have ⋃M∈{U∈Pow(Y). f-U∈T} by auto
  }
  moreover
  {
```

```
    fix U V assume U ∈ {U∈Pow(Y). f-U∈T} V∈{U∈Pow(Y). f-U∈T}
    then have U∈Pow(Y) V∈Pow(Y) f-U∈Tf-V∈T by auto
    then have f-(U∩V)∈T using topSpaceAssum invim_inter_inter_invim assms
      unfolding IsATopology_def surj_def by auto
    with ‹U∈Pow(Y)› ‹V∈Pow(Y)› have U∩V∈{U∈Pow(Y). f-(U)∈T} by auto
  }
  ultimately show thesis using IsATopology_def by auto
qed
```

The quotient function is continuous.

```
lemma (in topology0) quotient_func_cont:
  assumes f∈surj(⋃T,Y)
  shows IsContinuous(T,({quotient topology in} Y {by} f),f)
    unfolding IsContinuous_def using QuotientTop_def assms by auto
```

One of the important properties of this topology, is that a function from the quotient space is continuous iff the composition with the quotient function is continuous.

```
theorem (in two_top_spaces0) cont_quotient_top:
  assumes h∈surj(⋃τ₁,Y) g:Y→⋃τ₂ IsContinuous(τ₁,τ₂,g O h)
  shows IsContinuous(({quotient topology in} Y {by} h {from} τ₁),τ₂,g)
proof-
  {
    fix U assume U∈τ₂
    with assms(3) have (g O h)-(U)∈τ₁ unfolding IsContinuous_def by auto
    then have h-(g-(U))∈τ₁ using vimage_comp by auto
    with assms(1) have g-(U)∈({quotient topology in} Y {by} h {from}
τ₁)
      using topology0.QuotientTop_def tau1_is_top func1_1_L3 assms(2)

      unfolding topology0_def by auto
  }
  then show thesis unfolding IsContinuous_def by auto
qed
```

The underlying set of the quotient topology is Y.

```
lemma (in topology0) total_quo_func:
  assumes f∈surj(⋃T,Y)
  shows (⋃({quotient topology in} Y {by} f))=Y
proof-
  from assms have f-Y=⋃T using func1_1_L4 unfolding surj_def by auto
moreover
  have ⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto ul-
timately
  have Y∈({quotient topology in}Y{by}f{from}T) using QuotientTop_def
assms by auto
  then show thesis using QuotientTop_def assms by auto
qed
```

84.2 Quotient topologies from equivalence relations

In this section we will show that the quotient topologies come from an equivalence relation.

The quotient projection $b \mapsto r\{b\}$ is a function that maps the domain of the relation to the quotient. Note we do not need to assume that r is an equivalence relation.

```
lemma quotient_proj_fun:
  shows {⟨b,r{b}⟩. b∈A}:A→A//r unfolding Pi_def function_def domain_def
    unfolding quotient_def by auto
```

The quotient projection is a surjection. Again r does not need to be an equivalence relation here

```
lemma quotient_proj_surj:
  shows {⟨b,r{b}⟩. b∈A}∈surj(A,A//r)
proof-
  { fix y assume y∈A//r
    then obtain x where x∈A y=r{x} unfolding quotient_def by auto
    then have ∃x∈A. {⟨b,r{b}⟩. b∈A}(x) = y using ZF_fun_from_tot_val1

      by auto
  }
  then show thesis using quotient_proj_fun unfolding surj_def by auto
qed
```

The inverse image of a subset U of the quotient by the quotient projection is the union of U. Note since U is a subset of A/r it is a collection of equivalence classes.

```
lemma preim_equi_proj:
  assumes U⊆A//r equiv(A,r)
  shows {⟨b,r{b}⟩. b∈A}-(U) = ⋃U
proof
  {
    fix y assume y∈⋃U
    then obtain V where y∈V and V∈U by auto
    with assms have y ∈ {⟨b,r{b}⟩. b∈A}-(U)
      using EquivClass_1_L1 EquivClass_1_L2 by blast
  }
  thus ⋃U⊆{⟨b,r{b}⟩. b∈A}-U by blast
  {
    fix y assume y∈{⟨b,r{b}⟩. b∈A}-U
    then have yy: y∈{x∈A. r{x}∈U} by auto
    hence r{y}∈U by auto
    moreover from yy have y∈r{y} using assms equiv_class_self by auto

    ultimately have y∈⋃U by auto
  }
```

```
  thus {⟨b,r{b}⟩. b∈A}-U⊆⋃U by blast
qed
```

Now we define what a quotient topology from an equivalence relation is:

```
definition (in topology0)
  EquivQuo ({quotient by} _ 70)
  where equiv(⋃T,r)⟹({quotient by} r) ≡ {quotient topology in} (⋃T)//r
{by} {⟨b,r{b}⟩. b∈⋃T}
```

Outside of the topology0 context we need to indicate the original topology.

```
abbreviation EquivQuoTop (_{quotient by}_)
  where T {quotient by} r ≡ topology0.EquivQuo(T,r)
```

First, another description of the topology (more intuitive):

```
theorem (in topology0) quotient_equiv_rel:
  assumes equiv(⋃T,r)
  shows ({quotient by}r)={U∈Pow((⋃T)//r). ⋃U∈T}
proof-
  have ({quotient topology in}(⋃T)//r{by}{⟨b,r{b}⟩. b∈⋃T})=
    {U∈Pow((⋃T)//r). {⟨b,r{b}⟩. b∈⋃T}-U∈T}
    using QuotientTop_def quotient_proj_surj by auto
  moreover have {U∈Pow((⋃T)//r). {⟨b,r{b}⟩. b∈⋃T}-U∈T}={U∈Pow((⋃T)//r).
⋃U∈T}
  proof
    {
      fix U assume U∈{U∈Pow((⋃T)//r). {⟨b,r{b}⟩. b∈⋃T}-U∈T}
      with assms have U∈{U∈Pow((⋃T)//r). ⋃U∈T} using preim_equi_proj
by auto
    }
    thus {U∈Pow((⋃T)//r). {⟨b,r{b}⟩. b∈⋃T}-U∈T}⊆{U∈Pow((⋃T)//r). ⋃U∈T}
by auto
    {
      fix U assume U∈{U∈Pow((⋃T)//r). ⋃U∈T}
      with assms have U∈{U∈Pow((⋃T)//r). {⟨b,r{b}⟩. b∈⋃T}-U∈T} us-
ing preim_equi_proj
        by auto
     }
     thus {U∈Pow((⋃T)//r). ⋃U∈T}⊆{U∈Pow((⋃T)//r). {⟨b,r{b}⟩. b∈⋃T}-U∈T}
by auto
   qed
  ultimately show thesis using EquivQuo_def assms by auto
qed
```

We apply previous results to this topology.

```
theorem (in topology0) total_quo_equi:
  assumes equiv(⋃T,r)
  shows ⋃({quotient by}r)=(⋃T)//r
  using total_quo_func quotient_proj_surj EquivQuo_def assms by auto
```

The quotient by an equivalence relation is indeed a topology.

```
theorem (in topology0) equiv_quo_is_top:
  assumes equiv(⋃T,r)
  shows ({quotient by}r){is a topology}
  using quotientTop_is_top quotient_proj_surj EquivQuo_def assms by auto
```

The next theorem is the main result of this section: all quotient topologies arise from an equivalence relation given by the quotient function $f : X \to Y$. This means that any quotient topology is homeomorphic to a topology given by an equivalence relation quotient.

```
theorem (in topology0) equiv_quotient_top:
  assumes f∈surj(⋃T,Y)
  defines r≡{⟨x,y⟩∈⋃T×⋃T. f(x)=f(y)}
  defines g≡{⟨y,f-{y}⟩. y∈Y}
  shows equiv(⋃T,r) and
    IsAhomeomorphism(({quotient topology in}Y{by}f),({quotient by}r),g)
proof-
  from assms(1) have ff: f:⋃T→Y unfolding surj_def by auto
  from assms(2) show B: equiv(⋃T,r)
    unfolding equiv_def refl_def sym_def trans_def r_def by auto
  have gg: g:Y→((⋃T)//r)
  proof -
    { fix B assume B∈g
      then obtain y where Y:y∈Y B=⟨y,f-{y}⟩ unfolding g_def
        by auto
      then have f-{y}⊆⋃T using func1_1_L3 ff by blast
      then have eq: f-{y}={x∈⋃T. ⟨x,y⟩∈f} using vimage_iff by auto
      from assms(1) Y obtain A where A1: A∈⋃T f(A)=y unfolding surj_def

        by blast
      with ff have A: A ∈ f-{y} using func1_1_L15 by simp
      { fix t assume t∈f-{y}
        with A eq have t∈⋃T A∈⋃T ⟨t,y⟩∈f ⟨A,y⟩∈f by auto
        then have ft=fA using apply_equality assms(1) unfolding surj_def
by auto
        with assms(2) ‹t∈⋃T›‹A∈⋃T› have ⟨A,t⟩∈r by auto
        then have t∈r{A} using image_iff by auto
      } hence f-{y}⊆r{A} by auto
      moreover
      { fix t assume t∈r{A}
        with assms(2) have un: t∈⋃T A∈⋃T and eq2: f(t)=f(A)
          using image_iff by simp_all
        from ff un have ⟨t,f(t)⟩∈f using func1_1_L5A(1) by simp
        with eq2 A1 un eq have t∈f-{y} by simp
      } hence r{A}⊆f-{y} by auto
      ultimately have f-{y}=r{A} by auto
      with A1(1) have f-{y} ∈ (⋃T)//r
        using A1(1) unfolding quotient_def by auto
      with Y have B∈Y×(⋃T)//r by auto
```

```
    } then show thesis unfolding Pi_def function_def domain_def g_def

      by auto
  qed
  then have gg2: g:Y→(⋃({quotient by}r)) using total_quo_equi B
    by auto
  { fix s assume S: s∈({quotient topology in}Y{by}f)
    then have s∈Pow(Y) and P: f-(s)∈T
      using QuotientTop_def topSpaceAssum assms(1) by auto
    have f-s=(⋃y∈s. f-{y}) using vimage_eq_UN by blast
    moreover
    from ‹s∈Pow(Y)› have ∀y∈s. ⟨y,f-{y}⟩∈g unfolding g_def by auto
    then have ∀y∈s. gy=f-{y} using apply_equality gg by auto
    ultimately have f-s=(⋃y∈s. gy) by auto
    with P have (⋃y∈s. gy)∈T by auto
    moreover from ‹s∈Pow(Y)› have ∀y∈s. gy∈(⋃T)//r using apply_type
gg by auto
    ultimately have {gy. y∈s}∈({quotient by}r) using quotient_equiv_rel
B by auto
    with ‹s∈Pow(Y)› have gs∈({quotient by}r) using func_imagedef gg by
auto
  } hence gopen: ∀s∈({quotient topology in}Y{by}f). gs∈(T{quotient by}r)

    by auto
  have pr_fun: {⟨b,r{b}⟩. b∈⋃T}:⋃T→(⋃T)//r
    using quotient_proj_fun by auto
  { fix b assume b: b∈⋃T
    have bY: f(b)∈Y using apply_funtype ff b by auto
    with b have com: (g O f)b=g(fb) using comp_fun_apply ff by auto
    from bY have pg: ⟨fb,f-({fb})⟩∈g unfolding g_def by auto
    then have g(fb) = f-({fb}) using apply_equality gg by auto
    with com have comeq: (g O f)b=f-({fb}) by auto
    from b have A: f{b}={fb} {b}⊆⋃T using func_imagedef ff by auto
    from A(2) have b ∈ f-(f  {b}) using func1_1_L9 ff by blast
    with A(1) have b∈f-({fb}) by auto
    moreover from pg have f-({fb})∈(⋃T)//r using gg unfolding Pi_def

      by auto
    ultimately have r{b}=f-({fb}) using EquivClass_1_L2 B by auto
    then have (g O f)b=r{b} using comeq by auto
    moreover from b pr_fun have {⟨b,r{b}⟩. b∈⋃T}b=r{b}
      using apply_equality by simp
    ultimately  have (g O f)b={⟨b,r{b}⟩. b∈⋃T}b by simp
  } hence reg: ∀b∈⋃T. (g O f)(b)={⟨b,r{b}⟩. b∈⋃T}(b) by blast
  moreover have compp: g O f∈⋃T→(⋃T)//r using comp_fun ff gg by blast
  moreover
  from compp pr_fun reg have feq: (g O f)={⟨b,r{b}⟩. b∈⋃T}
    by (rule func_eq)
  then have IsContinuous(T,{quotient by}r,(g O f))
```

```
    using quotient_func_cont quotient_proj_surj EquivQuo_def topSpaceAssum
B by auto
  moreover have (g O f):⋃T→⋃({quotient by}r) using comp_fun ff gg2
by auto
  ultimately have gcont: IsContinuous({quotient topology in}Y{by}f,{quotient
by}r,g)
    using two_top_spaces0.cont_quotient_top assms(1) gg2 topSpaceAssum
equiv_quo_is_top B
    unfolding two_top_spaces0_def by auto
  { fix x y assume T: x∈Y y∈Y g(x)=g(y)
    with assms(3) have f(f-{x})=f(f-{y})
      using apply_equality gg by simp
    with assms(1) T(1,2) have x=y using surj_image_vimage by auto
  } with gg2 have g∈inj(Y,⋃({quotient by}r)) unfolding inj_def by auto

  moreover
  have g O f∈surj(⋃T, (⋃T)//r) using feq quotient_proj_surj by auto
  with ff gg B have g∈surj(Y,⋃(T{quotient by}r))
    using comp_mem_surjD1 total_quo_equi by auto
  ultimately have g∈bij(⋃({quotient topology in}Y{by}f),⋃({quotient
by}r))
    unfolding bij_def using total_quo_func assms(1) by auto
  with gcont gopen show IsAhomeomorphism(({quotient topology in}Y{by}f),({quotient
by}r),g)
    using bij_cont_open_homeo by auto
qed
```

The mapping $\langle b,c\rangle \mapsto \langle r\{a\}, r\{b\}\rangle$ is a function that maps the product of the carrier by itself to the product of the quotients. Note r does not have to be an equivalence relation.

```
lemma product_equiv_rel_fun:
  shows {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}:(⋃T×⋃T)→((⋃T)//r×(⋃T)//r)
proof-
  have  {⟨b,r{b}⟩. b∈⋃T}∈⋃T→(⋃T)//r using quotient_proj_fun by auto

  moreover have ∀A∈⋃T. ⟨A,r{A}⟩∈{⟨b,r{b}⟩. b∈⋃T} by auto
  ultimately have ∀A∈⋃T. {⟨b,r{b}⟩. b∈⋃T}A=r{A} using apply_equality
by auto
  hence {⟨⟨b,c⟩,r{b},r{c}⟩. ⟨b,c⟩∈⋃T×⋃T} =
    {⟨⟨x,y⟩,{⟨b,r{b}⟩. b∈⋃T}(x),{⟨b,r{b}⟩. b∈⋃T}(y)⟩. ⟨x,y⟩∈⋃T×⋃T}
    by force
  then show thesis using prod_fun quotient_proj_fun by auto
qed
```

The mapping $\langle b,c\rangle \mapsto \langle r\{a\}, r\{b\}\rangle$ is a surjection of the product of the carrier by itself onto the carrier of the product topology. Again r does not have to be an equivalence relation for this.

```
lemma (in topology0) prod_equiv_rel_surj:
  shows {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}∈surj(⋃(T×tT),((⋃T)//r×(⋃T)//r))
```

```
proof-
  have
    fun: {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}:(⋃T×⋃T)→((⋃T)//r×(⋃T)//r)

      using product_equiv_rel_fun by auto
  moreover
  { fix M assume M∈((⋃T)//r×(⋃T)//r)
    then obtain M1 M2 where M: M=⟨M1,M2⟩ M1∈(⋃T)//rM2∈(⋃T)//r by auto
    then obtain m1 m2 where m: m1∈⋃T m2∈⋃T M1=r{m1} M2=r{m2}
      unfolding quotient_def by auto
    then have mm: ⟨m1,m2⟩∈(⋃T×⋃T) by auto
    hence ⟨⟨m1,m2⟩,⟨r{m1},r{m2}⟩⟩∈{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}
      by auto
    with fun have {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}⟨m1,m2⟩=⟨r{m1},r{m2}⟩
      using apply_equality by auto
    with M(1) m(3,4) mm have ∃R∈(⋃T×⋃T). {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}(R)
= M
      by auto
  }
  ultimately show thesis unfolding surj_def using Top_1_4_T1(3) topSpaceAssum

    by auto
qed
```

The product quotient projection (i.e. the mapping the mapping $\langle b,c\rangle \mapsto \langle r\{a\}, r\{b\}\rangle$ is continuous.

```
lemma (in topology0) product_quo_fun:
  assumes equiv(⋃T,r)
  shows
    IsContinuous(T×tT,({quotient by} r)×t({quotient by} r),{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩.
⟨b,c⟩∈⋃T×⋃T})
proof-
  have {⟨b,r{b}⟩. b∈⋃T}:⋃T→(⋃T)//r using quotient_proj_fun by auto
  moreover have ∀A∈⋃T. ⟨A,r{A}⟩ ∈ {⟨b,r{b}⟩. b∈⋃T} by auto
  ultimately have ∀A∈⋃T. {⟨b,r{b}⟩. b∈⋃T}A=r{A} using apply_equality
by auto
  hence IN: {⟨⟨b,c⟩,r{b},r{c}⟩. ⟨b,c⟩∈⋃T×⋃T} =
    {⟨⟨x,y⟩,{⟨b,r{b}⟩. b∈⋃T}(x),{⟨b,r{b}⟩.b∈⋃T}(y)⟩. ⟨x,y⟩∈⋃T×⋃T}
    by force
  with assms have cont: IsContinuous(T,{quotient by}r,{⟨b,r{b}⟩. b∈⋃T})

    using quotient_func_cont quotient_proj_surj EquivQuo_def by auto
  with assms have  tot: ⋃(T{quotient by}r) = (⋃T) // r and
    top: ({quotient by}r) {is a topology}
    using total_quo_equi equiv_quo_is_top by auto
  then have fun: {⟨b,r{b}⟩. b∈⋃T}:⋃T→⋃({quotient by}r) using quotient_proj_fun

    by auto
  then have two_top_spaces0(T,{quotient by}r,{⟨b,r{b}⟩. b∈⋃T})
```

```
    unfolding two_top_spaces0_def using topSpaceAssum top by auto
  with fun cont top IN show thesis
    using two_top_spaces0.product_cont_functions topSpaceAssum by auto
qed
```

The product of quotient topologies is a quotient topology given that the quotient map is open. This isn't true in general.

```
theorem (in topology0) prod_quotient:
  assumes equiv(⋃T,r) ∀A∈T. {⟨b,r{b}⟩. b∈⋃T}(A) ∈ ({quotient by} r)
  shows (({quotient by} r)×t{quotient by} r) =
  ({quotient topology in} (((⋃T)//r)×((⋃T)//r)) {by} ({⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩.
⟨b,c⟩∈⋃T×⋃T}) {from} (T×tT))
proof
  let Tr = {quotient by} r
  let R = ({quotient topology in} (((⋃T)//r)×((⋃T)//r)) {by} ({⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩.
⟨b,c⟩∈⋃T×⋃T}) {from} (T×tT))
  { fix A assume A: A∈Tr×tTr
    with assms(1) have {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}-(A) ∈ T×tT
      using product_quo_fun unfolding IsContinuous_def by auto
    moreover
    from A have A⊆⋃((Tr)×t(Tr)) by auto
    with assms(1) have A∈Pow(((⋃T)//r)×((⋃T)//r))
      using Top_1_4_T1(3) equiv_quo_is_top total_quo_equi by auto
    ultimately have A∈R
      using topology0.QuotientTop_def Top_1_4_T1(1) topSpaceAssum prod_equiv_rel_surj

      unfolding topology0_def by auto
  } thus (Tr)×t(Tr) ⊆ R by auto
  { fix A assume A∈R
    with assms(1) have
      A: A ⊆ ((⋃T)//r)×((⋃T)//r) {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}-(A)
∈ T×tT
      using topology0.QuotientTop_def Top_1_4_T1(1) topSpaceAssum prod_equiv_rel_surj

      unfolding topology0_def by auto
    { fix C assume C∈A
      with A(1) obtain C1 C2 where CC: C=⟨C1,C2⟩ C1∈((⋃T)//r) C2∈((⋃T)//r)
by auto
      then obtain c1 c2 where CC1: c1∈⋃T c2∈⋃T and CC2: C1=r{c1} C2=r{c2}

        unfolding quotient_def by auto
      with ‹C∈A› CC have ⟨c1,c2⟩∈{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}-(A)
        using vimage_iff by auto
      with A(2) have
        ∃V W. V∈T ∧ W∈T ∧ V×W ⊆ {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}-(A)
∧ ⟨c1,c2⟩ ∈ V×W
         using prod_top_point_neighb topSpaceAssum by blast
       then obtain V W where
         VW: V∈T W∈T V×W ⊆ {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}-(A) c1∈V
```

```
c₂∈W
         by blast
       let Vᵣ = {⟨b,r{b}⟩. b∈⋃T}(V)
       let Wᵣ = {⟨b,r{b}⟩. b∈⋃T}(W)
       from VW assms have P: (Vᵣ×Wᵣ) ∈ (Tᵣ)×ₜ(Tᵣ)
          using prod_open_open_prod equiv_quo_is_top by auto
       { fix S assume S∈(Vᵣ×Wᵣ)
         then obtain s₁ s₂ where S: S=⟨s₁,s₂⟩ s₁∈Vᵣ s₂∈Wᵣ
           by blast
         then obtain t₁ t₂ where
           T: ⟨t₁,s₁⟩∈{⟨b,r{b}⟩. b∈⋃T} ⟨t₂,s₂⟩∈{⟨b,r{b}⟩. b∈⋃T} t₁∈V t₂∈W

           using image_iff by auto
         with VW(3) have ∃S₀∈A. ⟨⟨t₁,t₂⟩,S₀⟩∈{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}

           using vimage_iff by auto
         then obtain S₀ where S₀∈A and
           ⟨⟨t₁,t₂⟩,S₀⟩∈{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}
           by auto
         moreover from S(1) T VW(1,2) have
           ⟨⟨t₁,t₂⟩,S⟩ ∈ {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}
            by auto
         ultimately have S∈A
            using product_equiv_rel_fun unfolding Pi_def function_def
            by auto
        } hence sub: Vᵣ×Wᵣ ⊆ A by blast
        from CC CC2 CC1 ‹c₁∈V› ‹c₂∈W› have C ∈ Vᵣ×Wᵣ
          using image_iff by auto
        with P sub have ∃U∈(Tᵣ)×ₜ(Tᵣ). U⊆A ∧ C∈U
          by (rule witness_exists1)
      } hence ∀C∈A. ∃U∈(Tᵣ)×ₜ(Tᵣ). C∈U ∧ U⊆A
          by blast
      with assms(1) have A∈(Tᵣ)×ₜ(Tᵣ)
        using topology0.open_neigh_open Top_1_4_T1 equiv_quo_is_top assms
        unfolding topology0_def by auto
  } thus R ⊆ (Tᵣ)×ₜ(Tᵣ) by auto
qed

end
```

85 Topology 9

```
theory Topology_ZF_9
imports Topology_ZF_2 Group_ZF_2 Topology_ZF_7 Topology_ZF_8
begin
```

85.1 Group of homeomorphisms

This theory file deals with the fact the set homeomorphisms of a topological space into itself forms a group.

First, we define the set of homeomorphisms.

definition
  HomeoG(T) ≡ {f:⋃T→⋃T. IsAhomeomorphism(T,T,f)}

The homeomorphisms are closed by composition.

```
lemma (in topology0) homeo_composition:
  assumes f∈HomeoG(T)g∈HomeoG(T)
  shows Composition(⋃T)⟨f, g⟩∈HomeoG(T)
proof-
  from assms have fun:f∈⋃T→⋃Tg∈⋃T→⋃T and homeo:IsAhomeomorphism(T,T,f)IsAhomeomorphi
unfolding HomeoG_def
    by auto
  from fun have f O g∈⋃T→⋃T using comp_fun by auto moreover
  from homeo have bij:f∈bij(⋃T,⋃T)g∈bij(⋃T,⋃T) and cont:IsContinuous(T,T,f)IsContinuou
and contconv:
    IsContinuous(T,T,converse(f))IsContinuous(T,T,converse(g)) unfold-
ing IsAhomeomorphism_def by auto
  from bij have f O g∈bij(⋃T,⋃T) using comp_bij by auto moreover
  from cont have IsContinuous(T,T,f O g) using comp_cont by auto more-
over
  have converse(f O g)=converse(g) O converse(f) using converse_comp by
auto
  with contconv have IsContinuous(T,T,converse(f O g)) using comp_cont
by auto ultimately
  have f O g∈HomeoG(T) unfolding HomeoG_def IsAhomeomorphism_def by auto
  then show thesis using func_ZF_5_L2 fun by auto
qed
```

The identity function is a homeomorphism.

```
lemma (in topology0) homeo_id:
  shows id(⋃T)∈HomeoG(T)
proof-
  have converse(id(⋃T)) O id(⋃T)=id(⋃T) using left_comp_inverse id_bij
by auto
  then have converse(id(⋃T))=id(⋃T) using right_comp_id by auto
  then show thesis unfolding HomeoG_def IsAhomeomorphism_def using id_cont
id_type id_bij
    by auto
qed
```

The homeomorphisms form a monoid and its neutral element is the identity.

```
theorem (in topology0) homeo_submonoid:
  shows IsAmonoid(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))
```

```
  TheNeutralElement(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))=id(⋃T)
proof-
  have cl:HomeoG(T) {is closed under} Composition(⋃T) unfolding IsOpClosed_def
using homeo_composition by auto
  moreover have sub:HomeoG(T)⊆⋃T→⋃T unfolding HomeoG_def by auto
moreover
  have ne:TheNeutralElement(⋃T→⋃T, Composition(⋃T))∈HomeoG(T) us-
ing homeo_id Group_ZF_2_5_L2(2) by auto
  ultimately show IsAmonoid(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))
using Group_ZF_2_5_L2(1)
    monoid0.group0_1_T1 unfolding monoid0_def by force
  from cl sub ne have TheNeutralElement(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×Home
Composition(⋃T))
    using Group_ZF_2_5_L2(1) group0_1_L6 by blast moreover
  have id(⋃T)=TheNeutralElement(⋃T→⋃T, Composition(⋃T)) using Group_ZF_2_5_L2(2)
by auto
  ultimately show TheNeutralElement(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T
by auto
qed
```

The homeomorphisms form a group, with the composition.

```
theorem(in topology0) homeo_group:
  shows IsAgroup(HomeoG(T),restrict(Composition(⋃T),HomeoG(T)×HomeoG(T)))
proof-
  {
    fix x assume AS:x∈HomeoG(T)
    then have surj:x∈surj(⋃T,⋃T) and bij:x∈bij(⋃T,⋃T) unfolding HomeoG_def
IsAhomeomorphism_def bij_def by auto
    from bij have converse(x)∈bij(⋃T,⋃T) using bij_converse_bij by
auto
    with bij have conx_fun:converse(x)∈⋃T→⋃Tx∈⋃T→⋃T unfolding bij_def
inj_def by auto
    from surj have id:x O converse(x)=id(⋃T) using right_comp_inverse
by auto
    from conx_fun have Composition(⋃T)⟨x,converse(x)⟩=x O converse(x)
using func_ZF_5_L2 by auto
    with id have Composition(⋃T)⟨x,converse(x)⟩=id(⋃T) by auto
    moreover have converse(x)∈HomeoG(T) unfolding HomeoG_def using conx_fun(1)
homeo_inv AS unfolding HomeoG_def
      by auto
    ultimately have ∃M∈HomeoG(T). Composition(⋃T)⟨x,M⟩=id(⋃T) by auto
  }
  then have ∀x∈HomeoG(T). ∃M∈HomeoG(T). Composition(⋃T)⟨x,M⟩=id(⋃T)
by auto
  then show thesis using homeo_submonoid definition_of_group by auto
qed
```

85.2 Examples computed

As a first example, we show that the group of homeomorphisms of the cocardinal topology is the group of bijective functions.

```
theorem homeo_cocardinal:
  assumes InfCard(Q)
  shows HomeoG(CoCardinal(X,Q))=bij(X,X)
proof
  from assms have n:Q≠0 unfolding InfCard_def by auto
  then show HomeoG(CoCardinal(X,Q)) ⊆ bij(X, X) unfolding HomeoG_def
IsAhomeomorphism_def
    using union_cocardinal by auto
  {
    fix f assume a:f∈bij(X,X)
    then have converse(f)∈bij(X,X) using bij_converse_bij by auto
    then have cinj:converse(f)∈inj(X,X) unfolding bij_def by auto
    from a have fun:f∈X→X unfolding bij_def inj_def by auto
    then have two:two_top_spaces0((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
unfolding two_top_spaces0_def
      using union_cocardinal assms n CoCar_is_topology by auto
    {
      fix N assume N{is closed in}(CoCardinal(X,Q))
      then have N_def:N=X ∨ (N∈Pow(X) ∧ N≺Q) using closed_sets_cocardinal
n by auto
      then have restrict(converse(f),N)∈bij(N,converse(f)N) using cinj
restrict_bij by auto
      then have N≈f-N unfolding vimage_def eqpoll_def by auto
      then have f-N≈N using eqpoll_sym by auto
      with N_def have N=X ∨ (f-N≺Q ∧ N∈Pow(X)) using eq_lesspoll_trans
by auto
      with fun have f-N=X ∨ (f-N≺Q ∧ (f-N)∈Pow(X)) using func1_1_L3
func1_1_L4 by auto
      then have f-N {is closed in}(CoCardinal(X,Q)) using closed_sets_cocardinal
n by auto
    }
    then have ∀N. N{is closed in}(CoCardinal(X,Q)) ⟶ f-N {is closed
in}(CoCardinal(X,Q)) by auto
    then have IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),f) us-
ing two_top_spaces0.Top_ZF_2_1_L4
      two_top_spaces0.Top_ZF_2_1_L3 two_top_spaces0.Top_ZF_2_1_L2 two
by auto
  }
  then have ∀f∈bij(X,X). IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
by auto
  then have ∀f∈bij(X,X). IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
∧ IsContinuous((CoCardinal(X,Q)),(CoCardinal(X,Q)),converse(f))
    using bij_converse_bij by auto
  then have ∀f∈bij(X,X). IsAhomeomorphism((CoCardinal(X,Q)),(CoCardinal(X,Q)),f)
unfolding IsAhomeomorphism_def
```

```
    using n union_cocardinal by auto
  then show bij(X,X)⊆HomeoG((CoCardinal(X,Q))) unfolding HomeoG_def bij_def
inj_def using n union_cocardinal
    by auto
qed
```

The group of homeomorphism of the excluded set is a direct product of the bijections on $X \setminus T$ and the bijections on $X \cap T$.

```
theorem homeo_excluded:
  shows HomeoG(ExcludedSet(X,T))={f∈bij(X,X). f(X-T)=(X-T)}
proof
  have sub1:X-T⊆X by auto
  {
    fix g assume g∈HomeoG(ExcludedSet(X,T))
    then have fun:g:X→X and bij:g∈bij(X,X) and hom:IsAhomeomorphism((ExcludedSet(X,T)),(E
unfolding HomeoG_def
      using union_excludedset unfolding IsAhomeomorphism_def by auto
    {
      assume A:g(X-T)=X and B:X∩T≠0
      have rfun:restrict(g,X-T):X-T→X using fun restrict_fun sub1 by
auto moreover
      from A fun have {gaa. aa∈X-T}=X using func_imagedef sub1 by auto
      then have ∀x∈X. x∈{gaa. aa∈X-T} by auto
      then have ∀x∈X. ∃aa∈X-T. x=gaa by auto
      then have ∀x∈X. ∃aa∈X-T. x=restrict(g,X-T)aa by auto
      with A have surj:restrict(g,X-T)∈surj(X-T,X) using rfun unfold-
ing surj_def by auto
      from B obtain d where d∈Xd∈T by auto
      with bij have gd∈X using apply_funtype unfolding bij_def inj_def
by auto
      then obtain s where restrict(g,X-T)s=gds∈X-T using surj unfold-
ing surj_def by blast
      then have gs=gd by auto
      with ‹d∈X›‹s∈X-T› have s=d using bij unfolding bij_def inj_def
by auto
      then have False using ‹s∈X-T› ‹d∈T› by auto
    }
    then have g(X-T)=X ⟶ X∩T=0 by auto
    then have reg:g(X-T)=X ⟶ X-T=X by auto
    then have g(X-T)=X ⟶ g(X-T)=X-T by auto
    then have g(X-T)=X ⟶ g∈{f∈bij(X,X). f(X-T)=(X-T)} using bij by
auto moreover
    {
      fix gg
      assume A:gg(X-T)≠X and hom2:IsAhomeomorphism((ExcludedSet(X,T)),(ExcludedSet(X,T)),g
      from hom2 have fun:gg∈X→X and bij:gg∈bij(X,X) unfolding IsAhomeomorphism_def
bij_def inj_def using union_excludedset by auto
      have sub:X-T⊆⋃(ExcludedSet(X,T)) using union_excludedset by auto
      with hom2 have gg(Interior(X-T,(ExcludedSet(X,T))))=Interior(gg(X-T),(ExcludedSet(X,T
```

```
        using int_top_invariant by auto moreover
      from sub1 have Interior(X-T,(ExcludedSet(X,T)))=X-T using interior_set_excludedset
by auto
      ultimately have gg(X-T)=Interior(gg(X-T),(ExcludedSet(X,T))) by
auto moreover
      have ss:gg(X-T)⊆X using fun func1_1_L6(2) by auto
      then have Interior(gg(X-T),(ExcludedSet(X,T))) = (gg(X-T))-T us-
ing interior_set_excludedset A
        by auto
      ultimately have eq:gg(X-T)=(gg(X-T))-T by auto
      {
        assume (gg(X-T))∩T≠0
        then obtain t where t∈T and im:t∈gg(X-T) by blast
        then have t∉(gg(X-T))-T by auto
        then have False using eq im by auto
      }
      then have (gg(X-T))∩T=0 by auto
      then have gg(X-T)⊆X-T using ss by blast
    }
    then have ∀gg. gg(X-T)≠X ∧ IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),gg)⟶
gg(X-T)⊆X-T by auto moreover
    from bij have conbij:converse(g)∈bij(X,X) using bij_converse_bij
by auto
    then have confun:converse(g)∈X→X unfolding bij_def inj_def by auto
    {
      assume A:converse(g)(X-T)=X and B:X∩T≠0
      have rfun:restrict(converse(g),X-T):X-T→X using confun restrict_fun
sub1 by auto moreover
      from A confun have {converse(g)aa. aa∈X-T}=X using func_imagedef
sub1 by auto
      then have ∀x∈X. x∈{converse(g)aa. aa∈X-T} by auto
      then have ∀x∈X. ∃aa∈X-T. x=converse(g)aa by auto
      then have ∀x∈X. ∃aa∈X-T. x=restrict(converse(g),X-T)aa by auto
      with A have surj:restrict(converse(g),X-T)∈surj(X-T,X) using rfun
unfolding surj_def by auto
      from B obtain d where d∈Xd∈T by auto
      with conbij have converse(g)d∈X using apply_funtype unfolding bij_def
inj_def by auto
      then obtain s where restrict(converse(g),X-T)s=converse(g)ds∈X-T
using surj unfolding surj_def by blast
      then have converse(g)s=converse(g)d by auto
      with ‹d∈X›‹s∈X-T› have s=d using conbij unfolding bij_def inj_def
by auto
      then have False using ‹s∈X-T› ‹d∈T› by auto
    }
    then have converse(g)(X-T)=X ⟶ X∩T=0 by auto
    then have converse(g)(X-T)=X ⟶ X-T=X by auto
    then have converse(g)(X-T)=X ⟶ g-(X-T)=(X-T) unfolding vimage_def
by auto
```

```
    then have G:converse(g)(X-T)=X ⟶ g(g-(X-T))=g(X-T) by auto
    have GG:g(g-(X-T))=(X-T) using sub1 surj_image_vimage bij unfold-
ing bij_def by auto
    with G have converse(g)(X-T)=X ⟶ g(X-T)=X-T by auto
    then have converse(g)(X-T)=X ⟶ g∈{f∈bij(X,X). f(X-T)=(X-T)} us-
ing bij by auto moreover
    from hom have IsAhomeomorphism(ExcludedSet(X,T), ExcludedSet(X,T),
converse(g)) using homeo_inv by auto
    moreover note hom ultimately have g∈{f∈bij(X,X). f(X-T)=(X-T)} ∨
(g(X-T)⊆X-T ∧ converse(g)(X-T)⊆X-T)
      by force
    then have g∈{f∈bij(X,X). f(X-T)=(X-T)} ∨ (g(X-T)⊆X-T ∧ g-(X-T)⊆X-T)
unfolding vimage_def by auto moreover
    have g-(X-T)⊆X-T ⟶ g(g-(X-T))⊆g(X-T) using func1_1_L8 by auto
    with GG have g-(X-T)⊆X-T ⟶ (X-T)⊆g(X-T) by force
    ultimately have g∈{f∈bij(X,X). f(X-T)=(X-T)} ∨ (g(X-T)⊆X-T ∧ (X-T)⊆g(X-T))
by auto
    then have g∈{f∈bij(X,X). f(X-T)=(X-T)} using bij by auto
  }
  then show HomeoG(ExcludedSet(X,T))⊆{f∈bij(X,X). f(X-T)=(X-T)} by auto
  {
    fix g assume as:g∈bij(X,X)g(X-T)=X-T
    then have inj:g∈inj(X,X) and im:g-(g(X-T))=g-(X-T) unfolding bij_def
by auto
    from inj have g-(g(X-T))=X-T using inj_vimage_image sub1 by force
    with im have as_3:g-(X-T)=X-T by auto
    {
      fix A
      assume A∈(ExcludedSet(X,T))
      then have A=X∨A∩T=0 A⊆X unfolding ExcludedSet_def by auto
      then have A⊆X-T∨A=X by auto moreover
      {
        assume A=X
        with as(1) have gA=X using surj_range_image_domain unfolding bij_def
by auto
      }
      moreover
      {
        assume A⊆X-T
        then have gA⊆g(X-T) using func1_1_L8 by auto
        then have gA⊆(X-T) using as(2)  by auto
      }
      ultimately have gA⊆(X-T) ∨ gA=X by auto
      then have gA∈(ExcludedSet(X,T)) unfolding ExcludedSet_def by auto
    }
    then have ∀A∈(ExcludedSet(X,T)). gA∈(ExcludedSet(X,T)) by auto more-
over
    {
      fix A assume A∈(ExcludedSet(X,T))
```

```
      then have A=X∨A∩T=0 A⊆X unfolding ExcludedSet_def by auto
      then have A⊆X-T∨A=X by auto moreover
      {
        assume A=X
        with as(1) have g-A=X using func1_1_L4 unfolding bij_def inj_def
by auto
      }
      moreover
      {
        assume A⊆X-T
        then have g-A⊆g-(X-T) using func1_1_L8 by auto
        then have g-A⊆(X-T) using as_3 by auto
      }
      ultimately have g-A⊆(X-T) ∨ g-A=X by auto
      then have g-A∈(ExcludedSet(X,T)) unfolding ExcludedSet_def by auto
    }
    then have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),g) unfold-
ing IsContinuous_def by auto moreover
    note as(1) ultimately have IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),g)

      using union_excludedset bij_cont_open_homeo by auto
    with as(1) have g∈HomeoG(ExcludedSet(X,T)) unfolding bij_def inj_def
HomeoG_def using union_excludedset by auto
  }
  then show {f ∈ bij(X, X) . f  (X - T) = X - T} ⊆ HomeoG(ExcludedSet(X,T))
by auto
qed
```

We now give some lemmas that will help us compute HomeoG(IncludedSet(X,T)).

```
lemma cont_in_cont_ex:
  assumes IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) f:X→X T⊆X
  shows IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f)
proof-
  from assms(2,3) have two:two_top_spaces0(IncludedSet(X,T),IncludedSet(X,T),f)
using union_includedset includedset_is_topology
    unfolding two_top_spaces0_def by auto
  {
    fix A assume A∈(ExcludedSet(X,T))
    then have A∩T=0 ∨ A=XA⊆X unfolding ExcludedSet_def by auto
    then have A{is closed in}(IncludedSet(X,T)) using closed_sets_includedset
assms by auto
    then have f-A{is closed in}(IncludedSet(X,T)) using two_top_spaces0.TopZF_2_1_L1
assms(1)
      two assms includedset_is_topology by auto
    then have (f-A)∩T=0 ∨ f-A=Xf-A⊆X using closed_sets_includedset assms(1,3)
by auto
    then have f-A∈(ExcludedSet(X,T)) unfolding ExcludedSet_def by auto
  }
  then show IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) unfold-
```

```
ing IsContinuous_def by auto
qed

lemma cont_ex_cont_in:
  assumes IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) f:X→X T⊆X
  shows IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f)
proof-
  from assms(2) have two:two_top_spaces0(ExcludedSet(X,T),ExcludedSet(X,T),f)
using union_excludedset excludedset_is_topology
    unfolding two_top_spaces0_def by auto
  {
    fix A assume A∈(IncludedSet(X,T))
    then have T⊆A ∨ A=0A⊆X unfolding IncludedSet_def by auto
    then have A{is closed in}(ExcludedSet(X,T)) using closed_sets_excludedset
assms by auto
    then have f-A{is closed in}(ExcludedSet(X,T)) using two_top_spaces0.TopZF_2_1_L1
assms(1)
      two assms excludedset_is_topology by auto
    then have T⊆(f-A) ∨ f-A=0f-A⊆X using closed_sets_excludedset assms(1,3)
by auto
    then have f-A∈(IncludedSet(X,T)) unfolding IncludedSet_def by auto
  }
  then show IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) unfold-
ing IsContinuous_def by auto
qed
```

The previous lemmas imply that the group of homeomorphisms of the included set topology is the same as the one of the excluded set topology.

```
lemma homeo_included:
  assumes T⊆X
  shows HomeoG(IncludedSet(X,T))={f ∈ bij(X, X) . f  (X - T) = X - T}
proof-
  {
    fix f assume f∈HomeoG(IncludedSet(X,T))
    then have hom:IsAhomeomorphism(IncludedSet(X,T),IncludedSet(X,T),f)
and fun:f∈X→X and
      bij:f∈bij(X,X) unfolding HomeoG_def IsAhomeomorphism_def using union_includedset
assms by auto
    then have cont:IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f)
unfolding IsAhomeomorphism_def by auto
    then have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f) using
cont_in_cont_ex fun assms by auto moreover
    {
      from hom have cont1:IsContinuous(IncludedSet(X,T),IncludedSet(X,T),converse(f))
unfolding IsAhomeomorphism_def by auto moreover
      have converse(f):X→X using bij_converse_bij bij unfolding bij_def
inj_def by auto moreover
      note assms ultimately
      have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),converse(f))
```

```
using cont_in_cont_ex assms by auto
    }
    then have IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),converse(f))
by auto
    moreover note bij ultimately
    have IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),f) unfold-
ing IsAhomeomorphism_def
      using union_excludedset by auto
    with fun have f∈HomeoG(ExcludedSet(X,T)) unfolding HomeoG_def us-
ing union_excludedset by auto
  }
  then have HomeoG(IncludedSet(X,T))⊆HomeoG(ExcludedSet(X,T)) by auto
moreover
  {
    fix f assume f∈HomeoG(ExcludedSet(X,T))
    then have hom:IsAhomeomorphism(ExcludedSet(X,T),ExcludedSet(X,T),f)
and fun:f∈X→X and
      bij:f∈bij(X,X) unfolding HomeoG_def IsAhomeomorphism_def using union_excludedset
assms by auto
    then have cont:IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),f)
unfolding IsAhomeomorphism_def by auto
    then have IsContinuous(IncludedSet(X,T),IncludedSet(X,T),f) using
cont_ex_cont_in fun assms by auto moreover
    {
      from hom have cont1:IsContinuous(ExcludedSet(X,T),ExcludedSet(X,T),converse(f))
unfolding IsAhomeomorphism_def by auto moreover
      have converse(f):X→X using bij_converse_bij bij unfolding bij_def
inj_def by auto moreover
      note assms ultimately
      have IsContinuous(IncludedSet(X,T),IncludedSet(X,T),converse(f))
using cont_ex_cont_in assms by auto
    }
    then have IsContinuous(IncludedSet(X,T),IncludedSet(X,T),converse(f))
by auto
    moreover note bij ultimately
    have IsAhomeomorphism(IncludedSet(X,T),IncludedSet(X,T),f) unfold-
ing IsAhomeomorphism_def
      using union_includedset assms by auto
    with fun have f∈HomeoG(IncludedSet(X,T)) unfolding HomeoG_def us-
ing union_includedset assms by auto
  }
  then have HomeoG(ExcludedSet(X,T))⊆HomeoG(IncludedSet(X,T)) by auto
ultimately
  show thesis using homeo_excluded by auto
qed
```

Finally, let's compute part of the group of homeomorphisms of an order topology.

```
lemma homeo_order:
```

```
  assumes IsLinOrder(X,r)∃x y. x≠y∧x∈X∧y∈X
  shows ord_iso(X,r,X,r)⊆HomeoG(OrdTopology X r)
proof
  fix f assume f∈ord_iso(X,r,X,r)
  then have bij:f∈bij(X,X) and ord:∀x∈X. ∀y∈X. ⟨x, y⟩ ∈ r ⟷ ⟨f  x,
f  y⟩ ∈ r
    unfolding ord_iso_def by auto
  have twoSpac:two_top_spaces0(OrdTopology X r,OrdTopology X r,f) un-
folding two_top_spaces0_def
    using bij unfolding bij_def inj_def using union_ordtopology[OF assms]
Ordtopology_is_a_topology(1)[OF assms(1)]
    by auto
  {
    fix c d assume A:c∈Xd∈X
    {
      fix x assume AA:x∈Xx≠cx≠d⟨c,x⟩∈r⟨x,d⟩∈r
      then have ⟨fc,fx⟩∈r⟨fx,fd⟩∈r using A(2,1) ord by auto moreover
      {
        assume fx=fc ∨ fx=fd
        then have x=c∨x=d using bij unfolding bij_def inj_def using A(2,1)
AA(1) by auto
        then have False using AA(2,3) by auto
      }
      then have fx≠fcfx≠fd by auto moreover
      have fx∈X using bij unfolding bij_def inj_def using apply_type AA(1)
by auto
      ultimately have fx∈IntervalX(X,r,fc,fd) unfolding IntervalX_def
Interval_def by auto
    }
    then have {fx. x∈IntervalX(X,r,c,d)}⊆IntervalX(X,r,fc,fd) unfold-
ing IntervalX_def Interval_def by auto
    moreover
    {
      fix y assume y∈IntervalX(X,r,fc,fd)
      then have y:y∈Xy≠fcy≠fd⟨fc,y⟩∈r⟨y,fd⟩∈r unfolding IntervalX_def
Interval_def by auto
      then obtain s where s:s∈Xy=fs using bij unfolding bij_def surj_def
by auto
      {
        assume s=c∨s=d
        then have fs=fc∨fs=fd by auto
        then have False using s(2) y(2,3) by auto
      }
      then have s≠cs≠d by auto moreover
      have ⟨c,s⟩∈r⟨s,d⟩∈r using y(4,5) s ord A(2,1) by auto moreover
      note s(1) ultimately have s∈IntervalX(X,r,c,d) unfolding IntervalX_def
Interval_def by auto
      then have y∈{fx. x∈IntervalX(X,r,c,d)} using s(2) by auto
    }
```

```
    ultimately have {fx. x∈IntervalX(X,r,c,d)}=IntervalX(X,r,fc,fd) by
auto moreover
    have IntervalX(X,r,c,d)⊆X unfolding IntervalX_def by auto more-
over
    have f:X→X using bij unfolding bij_def surj_def by auto ultimately
    have fIntervalX(X,r,c,d)=IntervalX(X,r,fc,fd) using func_imagedef
by auto
  }
  then have inter:∀c∈X. ∀d∈X. fIntervalX(X,r,c,d)=IntervalX(X,r,fc,fd)
∧ fc∈X ∧ fd∈X using bij
    unfolding bij_def inj_def by auto
  {
    fix c assume A:c∈X
    {
      fix x assume AA:x∈Xx≠c⟨c,x⟩∈r
      then have ⟨fc,fx⟩∈r using A ord by auto moreover
      {
        assume fx=fc
        then have x=c using bij unfolding bij_def inj_def using A AA(1)
by auto
        then have False using AA(2) by auto
      }
      then have fx≠fc by auto moreover
      have fx∈X using bij unfolding bij_def inj_def using apply_type AA(1)
by auto
      ultimately have fx∈RightRayX(X,r,fc) unfolding RightRayX_def by
auto
    }
    then have {fx. x∈RightRayX(X,r,c)}⊆RightRayX(X,r,fc) unfolding RightRayX_def
by auto
    moreover
    {
      fix y assume y∈RightRayX(X,r,fc)
      then have y:y∈Xy≠fc⟨fc,y⟩∈r unfolding RightRayX_def by auto
      then obtain s where s:s∈Xy=fs using bij unfolding bij_def surj_def
by auto
      {
        assume s=c
        then have fs=fc by auto
        then have False using s(2) y(2) by auto
      }
      then have s≠c by auto moreover
      have ⟨c,s⟩∈r using y(3) s ord A by auto moreover
      note s(1) ultimately have s∈RightRayX(X,r,c) unfolding RightRayX_def
by auto
      then have y∈{fx. x∈RightRayX(X,r,c)} using s(2) by auto
    }
    ultimately have {fx. x∈RightRayX(X,r,c)}=RightRayX(X,r,fc) by auto
moreover
```

```
    have RightRayX(X,r,c)⊆X unfolding RightRayX_def by auto moreover
    have f:X→X using bij unfolding bij_def surj_def by auto ultimately
    have fRightRayX(X,r,c)=RightRayX(X,r,fc) using func_imagedef by auto
  }
  then have rray:∀c∈X. fRightRayX(X,r,c)=RightRayX(X,r,fc) ∧ fc∈X us-
ing bij
    unfolding bij_def inj_def by auto
  {
    fix c assume A:c∈X
    {
      fix x assume AA:x∈Xx≠c⟨x,c⟩∈r
      then have ⟨fx,fc⟩∈r using A ord by auto moreover
      {
        assume fx=fc
        then have x=c using bij unfolding bij_def inj_def using A AA(1)
by auto
        then have False using AA(2) by auto
      }
      then have fx≠fc by auto moreover
      have fx∈X using bij unfolding bij_def inj_def using apply_type AA(1)
by auto
      ultimately have fx∈LeftRayX(X,r,fc) unfolding LeftRayX_def by auto
    }
    then have {fx. x∈LeftRayX(X,r,c)}⊆LeftRayX(X,r,fc) unfolding LeftRayX_def
by auto
    moreover
    {
      fix y assume y∈LeftRayX(X,r,fc)
      then have y:y∈Xy≠fc⟨y,fc⟩∈r unfolding LeftRayX_def by auto
      then obtain s where s:s∈Xy=fs using bij unfolding bij_def surj_def
by auto
      {
        assume s=c
        then have fs=fc by auto
        then have False using s(2) y(2) by auto
      }
      then have s≠c by auto moreover
      have ⟨s,c⟩∈r using y(3) s ord A by auto moreover
      note s(1) ultimately have s∈LeftRayX(X,r,c) unfolding LeftRayX_def
by auto
      then have y∈{fx. x∈LeftRayX(X,r,c)} using s(2) by auto
    }
    ultimately have {fx. x∈LeftRayX(X,r,c)}=LeftRayX(X,r,fc) by auto
moreover
    have LeftRayX(X,r,c)⊆X unfolding LeftRayX_def by auto moreover
    have f:X→X using bij unfolding bij_def surj_def by auto ultimately
    have fLeftRayX(X,r,c)=LeftRayX(X,r,fc) using func_imagedef by auto
  }
  then have lray:∀c∈X. fLeftRayX(X,r,c)=LeftRayX(X,r,fc)∧fc∈X using
```

```
bij
    unfolding bij_def inj_def by auto
  have r1:∀U∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X, r,
b) . b ∈ X} ∪
    {RightRayX(X, r, b) . b ∈ X}. fU∈({IntervalX(X, r, b, c) . ⟨b,c⟩ ∈
X × X} ∪ {LeftRayX(X, r, b) . b ∈ X} ∪
    {RightRayX(X, r, b) . b ∈ X}) apply safe prefer 3 using rray apply
blast prefer 2 using lray apply blast
    using inter apply auto
    proof-
     fix xa y assume xa∈Xy∈X
     then have fxa∈Xfy∈X using bij unfolding bij_def inj_def by auto
     then show ∃x∈X. ∃ya∈X. IntervalX(X, r, f  xa, f  y) = IntervalX(X,
r, x, ya) by auto
    qed
  have r2:{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X, r, b)
. b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X}⊆(OrdTopology X r)
    using base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]]
by blast
  {
    fix U assume U∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X}
    with r1 have fU∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X}
      by auto
    with r2 have fU∈(OrdTopology X r) by blast
  }
  then have ∀U∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪
    {RightRayX(X, r, b) . b ∈ X}. fU∈(OrdTopology X r) by blast
  then have f_open:∀U∈(OrdTopology X r). fU∈(OrdTopology X r) using two_top_spaces0.base_i
twoSpac Ordtopology_is_a_topology(2)[OF assms(1)]]
    by auto
  {
    fix c d assume A:c∈Xd∈X
    then obtain cc dd where pre:fcc=cfdd=dcc∈Xdd∈X using bij unfold-
ing bij_def surj_def by blast
    with inter have f  IntervalX(X, r, cc, dd) = IntervalX(X, r,  c,
d) by auto
    then have f-(fIntervalX(X, r, cc, dd)) = f-(IntervalX(X, r,  c,  d))
by auto
    moreover
    have IntervalX(X, r, cc, dd)⊆X unfolding IntervalX_def by auto more-
over
    have f∈inj(X,X) using bij unfolding bij_def by auto ultimately
    have IntervalX(X, r, cc, dd)=f-IntervalX(X, r,  c,  d) using inj_vimage_image
by auto
    moreover
    from pre(3,4) have IntervalX(X, r, cc, dd)∈{IntervalX(X,r,e1,e2).
```

```
⟨e1,e2⟩∈X×X} by auto
    ultimately have f-IntervalX(X, r,  c,  d)∈(OrdTopology X r) using
      base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]] by
auto
  }
  then have inter:∀c∈X. ∀d∈X. f-IntervalX(X, r,  c,  d)∈(OrdTopology
X r) by auto
  {
    fix c assume A:c∈X
    then obtain cc where pre:fcc=ccc∈X using bij unfolding bij_def surj_def
by blast
    with rray have f  RightRayX(X, r, cc) = RightRayX(X, r,  c) by auto
    then have f-(fRightRayX(X, r, cc)) = f-(RightRayX(X, r,  c)) by auto

    moreover
    have RightRayX(X, r, cc)⊆X unfolding RightRayX_def by auto more-
over
    have f∈inj(X,X) using bij unfolding bij_def by auto ultimately
    have RightRayX(X, r, cc)=f-RightRayX(X, r,  c) using inj_vimage_image
by auto
    moreover
    from pre(2) have RightRayX(X, r, cc)∈{RightRayX(X,r,e2). e2∈X} by
auto
    ultimately have f-RightRayX(X, r,  c)∈(OrdTopology X r) using
      base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]] by
auto
  }
  then have rray:∀c∈X. f-RightRayX(X, r,  c)∈(OrdTopology X r) by auto
  {
    fix c assume A:c∈X
    then obtain cc where pre:fcc=ccc∈X using bij unfolding bij_def surj_def
by blast
    with lray have f  LeftRayX(X, r, cc) = LeftRayX(X, r,  c) by auto
    then have f-(fLeftRayX(X, r, cc)) = f-(LeftRayX(X, r,  c)) by auto

    moreover
    have LeftRayX(X, r, cc)⊆X unfolding LeftRayX_def by auto moreover
    have f∈inj(X,X) using bij unfolding bij_def by auto ultimately
    have LeftRayX(X, r, cc)=f-LeftRayX(X, r,  c) using inj_vimage_image
by auto
    moreover
    from pre(2) have LeftRayX(X, r, cc)∈{LeftRayX(X,r,e2). e2∈X} by
auto
    ultimately have f-LeftRayX(X, r,  c)∈(OrdTopology X r) using
      base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]] by
auto
  }
  then have lray:∀c∈X. f-LeftRayX(X, r,  c)∈(OrdTopology X r) by auto
  {
```

```
    fix U assume U∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X}
    with lray inter rray have f-U∈(OrdTopology X r) by auto
  }
  then have ∀U∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X}.
    f-U∈(OrdTopology X r) by blast
  then have fcont:IsContinuous(OrdTopology X r,OrdTopology X r,f) us-
ing two_top_spaces0.Top_ZF_2_1_L5[OF twoSpac
    Ordtopology_is_a_topology(2)[OF assms(1)]] by auto
  from fcont f_open bij have IsAhomeomorphism(OrdTopology X r,OrdTopology
X r,f) using bij_cont_open_homeo
    union_ordtopology[OF assms] by auto
  then show f∈HomeoG(OrdTopology X r) unfolding HomeoG_def using bij
union_ordtopology[OF assms]
    unfolding bij_def inj_def by auto
qed
```

This last example shows that order isomorphic sets give homeomorphic topological spaces.

85.3 Properties preserved by functions

The continuous image of a connected space is connected.

```
theorem (in two_top_spaces0) cont_image_conn:
  assumes IsContinuous(τ₁,τ₂,f) f∈surj(X₁,X₂) τ₁{is connected}
  shows τ₂{is connected}
proof-
  {
    fix U
    assume Uop:U∈τ₂ and Ucl:U{is closed in}τ₂
    from Uop assms(1) have f-U∈τ₁ unfolding IsContinuous_def by auto
moreover
    from Ucl assms(1) have  f-U{is closed in}τ₁ using TopZF_2_1_L1 by
auto ultimately
    have disj:f-U=0 ∨ f-U=⋃τ₁ using assms(3) unfolding IsConnected_def
by auto moreover
    {
      assume as:f-U≠0
      then have U≠0 using func1_1_L13 by auto
      from as disj have f-U=⋃τ₁ by auto
      then have f(f-U)=f(⋃τ₁) by auto moreover
      have U⊆⋃τ₂ using Uop by blast ultimately
      have U=f(⋃τ₁) using surj_image_vimage assms(2) Uop by force
      then have ⋃τ₂=U using surj_range_image_domain assms(2) by auto
    }
    moreover
    {
```

```
      assume as:U≠0
      from Uop have s:U⊆⋃τ2 by auto
      with as obtain u where uU:u∈U by auto
      with s have u∈⋃τ2 by auto
      with assms(2) obtain w where fw=uw∈⋃τ1 unfolding surj_def X1_def
X2_def by blast
      with uU have w∈f-U using func1_1_L15 assms(2) unfolding surj_def
by auto
      then have f-U≠0 by auto
    }
    ultimately have U=0∨U=⋃τ2 by auto
  }
  then show thesis unfolding IsConnected_def by auto
qed
```

Every continuous function from a space which has some property P and a space which has the property anti(P), given that this property is preserved by continuous functions, if follows that the range of the function is in the spectrum. Applied to connectedness, it follows that continuous functions from a connected space to a totally-disconnected one are constant.

```
corollary(in two_top_spaces0) cont_conn_tot_disc:
  assumes IsContinuous(τ1,τ2,f) τ1{is connected} τ2{is totally-disconnected}
f:X1→X2 X1≠0
  shows ∃q∈X2. ∀w∈X1. f(w)=q
proof-
  from assms(4) have surj:f∈surj(X1,range(f)) using fun_is_surj by auto
  have sub:range(f)⊆X2 using func1_1_L5B assms(4) by auto
  from assms(1) have cont:IsContinuous(τ1,τ2{restricted to}range(f),f)
using restr_image_cont range_image_domain
    assms(4) by auto
  have union:⋃(τ2{restricted to}range(f))=range(f) unfolding RestrictedTo_def
using sub by auto
  then have two_top_spaces0(τ1,τ2{restricted to}range(f),f) unfolding
two_top_spaces0_def
    using surj unfolding surj_def using tau1_is_top topology0.Top_1_L4
unfolding topology0_def using tau2_is_top
    by auto
  then have conn:(τ2{restricted to}range(f)){is connected} using two_top_spaces0.cont_image
surj assms(2) cont
    union by auto
  then have range(f){is in the spectrum of}IsConnected using assms(3)
sub unfolding IsTotDis_def antiProperty_def
    using union by auto
  then have range(f)≲1 using conn_spectrum by auto moreover
  from assms(5) have fX1≠0 using func1_1_L15A assms(4) by auto
  then have range(f)≠0 using range_image_domain assms(4) by auto
  ultimately obtain q where uniq:range(f)={q} using lepoll_1_is_sing by
blast
  {
```

```
    fix w assume w∈X₁
    then have fw∈range(f) using func1_1_L5A(2) assms(4) by auto
    with uniq have fw=q by auto
  }
  then have ∀w∈X₁. fw=q by auto
  then show thesis using uniq sub by auto
qed
```

The continuous image of a compact space is compact.

```
theorem (in two_top_spaces0) cont_image_com:
  assumes IsContinuous(τ₁,τ₂,f) f∈surj(X₁,X₂) X₁{is compact of cardinal}K{in}τ₁
  shows X₂{is compact of cardinal}K{in}τ₂
proof-
  have X₂⊆⋃τ₂ by auto moreover
  {
    fix U assume as:X₂⊆⋃U U⊆τ₂
    then have P:{f-V. V∈U}⊆τ₁ using assms(1) unfolding IsContinuous_def
by auto
    from as(1) have f-X₂ ⊆ f-(⋃U) by blast
    then have f-X₂ ⊆ converse(f)(⋃U) unfolding vimage_def by auto more-
over
    have converse(f)(⋃U)=(⋃V∈U. converse(f)V) using image_UN by force
ultimately
    have f-X₂ ⊆ (⋃V∈U. converse(f)V) by auto
    then have f-X₂ ⊆ (⋃V∈U. f-V) unfolding vimage_def by auto
    then have X₁ ⊆ (⋃V∈U. f-V) using func1_1_L4 assms(2) unfolding surj_def
by force
    then have X₁ ⊆ ⋃{f-V. V∈U} by auto
    with P assms(3) have ∃N∈Pow({f-V. V∈U}). X₁ ⊆ ⋃N ∧ N≺K unfold-
ing IsCompactOfCard_def by auto
    then obtain N where N∈Pow({f-V. V∈U}) X₁ ⊆ ⋃N N≺K by auto
    then have fin:N≺K and sub:N⊆{f-V. V∈U} and cov:X₁ ⊆ ⋃N unfold-
ing FinPow_def by auto
    from sub have {fR. R∈N}⊆{f(f-V). V∈U} by auto moreover
    have ∀V∈U. V⊆⋃τ₂ using as(2) by auto ultimately
    have {fR. R∈N}⊆U using surj_image_vimage assms(2) by auto more-
over
    let FN={⟨R,fR⟩. R∈N}
    have FN:FN:N→{fR. R∈N} unfolding Pi_def function_def domain_def by
auto
    {
      fix S assume S∈{fR. R∈N}
      then obtain R where R_def:R∈NfR=S by auto
      then have ⟨R,fR⟩∈FN by auto
      then have FNR=fR using FN apply_equality by auto
      then have ∃R∈N. FNR=S using R_def by auto
    }
    then have surj:FN∈surj(N,{fR. R∈N}) unfolding surj_def using FN by
force
```

```
    from fin have N:N≲K Ord(K) using assms(3) lesspoll_imp_lepoll unfolding IsCompactOfCard_def
      using Card_is_Ord by auto
    then have {fR. R∈N}≲N using surj_fun_inv_2 surj by auto
    then have {fR. R∈N}≺K using fin lesspoll_trans1 by blast
    moreover
    have ⋃{fR. R∈N}=f(⋃N) using image_UN by auto
    then have fX1 ⊆ ⋃{fR. R∈N} using cov by blast
    then have X2 ⊆ ⋃{fR. R∈N} using assms(2) surj_range_image_domain by auto
    ultimately have ∃NN∈Pow(U). X2 ⊆ ⋃NN ∧ NN≺K by auto
  }
  then have ∀U∈Pow(τ2). X2 ⊆ ⋃U ⟶ (∃NN∈Pow(U). X2 ⊆ ⋃NN ∧ NN≺K) by auto
  ultimately show thesis using assms(3) unfolding IsCompactOfCard_def by auto
qed
```

As it happends to connected spaces, a continuous function from a compact space to an anti-compact space has finite range.

```
corollary (in two_top_spaces0) cont_comp_anti_comp:
  assumes IsContinuous(τ1,τ2,f) X1{is compact in}τ1 τ2{is anti-compact} f:X1→X2 X1≠0
  shows Finite(range(f)) and range(f)≠0
proof-
  from assms(4) have surj:f∈surj(X1,range(f)) using fun_is_surj by auto
  have sub:range(f)⊆X2 using func1_1_L5B assms(4) by auto
  from assms(1) have cont:IsContinuous(τ1,τ2{restricted to}range(f),f) using restr_image_cont range_image_domain
    assms(4) by auto
  have union:⋃(τ2{restricted to}range(f))=range(f) unfolding RestrictedTo_def using sub by auto
  then have two_top_spaces0(τ1,τ2{restricted to}range(f),f) unfolding two_top_spaces0_def
    using surj unfolding surj_def using tau1_is_top topology0.Top_1_L4 unfolding topology0_def using tau2_is_top
    by auto
  then have range(f){is compact in}(τ2{restricted to}range(f)) using surj two_top_spaces0.cont_image_com cont union
    assms(2) Compact_is_card_nat by force
  then have range(f){is in the spectrum of}(λT. (⋃T) {is compact in}T) using assms(3) sub unfolding IsAntiComp_def antiProperty_def
    using union by auto
  then show Finite(range(f)) using compact_spectrum by auto moreover
  from assms(5) have fX1≠0 using func1_1_L15A assms(4) by auto
  then show range(f)≠0 using range_image_domain assms(4) by auto
qed
```

As a consequence, it follows that quotient topological spaces of compact (connected) spaces are compact (connected).

```
corollary(in topology0) compQuot:
  assumes (⋃T){is compact in}T equiv(⋃T,r)
  shows (⋃T)//r{is compact in}({quotient by}r)
proof-
  have surj:{⟨b,r{b}⟩. b∈⋃T}∈surj(⋃T,(⋃T)//r) using quotient_proj_surj
by auto
  moreover have tot:⋃({quotient by}r)=(⋃T)//r using total_quo_equi
assms(2) by auto
  ultimately have cont:IsContinuous(T,{quotient by}r,{⟨b,r{b}⟩. b∈⋃T})
using quotient_func_cont
    EquivQuo_def assms(2) by auto
  from surj tot have two_top_spaces0(T,{quotient by}r,{⟨b,r{b}⟩. b∈⋃T})
unfolding two_top_spaces0_def
    using topSpaceAssum equiv_quo_is_top assms(2) unfolding surj_def by
auto
  with surj cont tot assms(1) show thesis using two_top_spaces0.cont_image_com
Compact_is_card_nat by force
qed

corollary(in topology0) ConnQuot:
  assumes T{is connected} equiv(⋃T,r)
  shows ({quotient by}r){is connected}
proof-
  have surj:{⟨b,r{b}⟩. b∈⋃T}∈surj(⋃T,(⋃T)//r) using quotient_proj_surj
by auto
  moreover have tot:⋃({quotient by}r)=(⋃T)//r using total_quo_equi
assms(2) by auto
  ultimately have cont:IsContinuous(T,{quotient by}r,{⟨b,r{b}⟩. b∈⋃T})
using quotient_func_cont
    EquivQuo_def assms(2) by auto
  from surj tot have two_top_spaces0(T,{quotient by}r,{⟨b,r{b}⟩. b∈⋃T})
unfolding two_top_spaces0_def
    using topSpaceAssum equiv_quo_is_top assms(2) unfolding surj_def by
auto
  with surj cont tot assms(1) show thesis using two_top_spaces0.cont_image_conn
by force
qed

end
```

86 Topology 10

```
theory Topology_ZF_10
imports Topology_ZF_7
begin
```

This file deals with properties of product spaces. We only consider product of two spaces, and most of this proofs, can be used to prove the results in product of a finite number of spaces.

86.1 Closure and closed sets in product space

The closure of a product, is the product of the closures.

```
lemma cl_product:
  assumes T{is a topology} S{is a topology} A⊆⋃T B⊆⋃S
  shows Closure(A×B,ProductTopology(T,S))=Closure(A,T)×Closure(B,S)
proof
  have A×B⊆⋃T×⋃S using assms(3,4) by auto
  then have sub:A×B⊆⋃ProductTopology(T,S) using Top_1_4_T1(3) assms(1,2)
by auto
  have top:ProductTopology(T,S){is a topology} using Top_1_4_T1(1) assms(1,2)
by auto
  {
    fix x assume asx:x∈Closure(A×B,ProductTopology(T,S))
    then have reg:∀U∈ProductTopology(T,S). x∈U ⟶ U∩(A×B)≠0 using
topology0.cl_inter_neigh
      sub top unfolding topology0_def by blast
    from asx have x∈⋃ProductTopology(T,S) using topology0.Top_3_L11(1)
top unfolding topology0_def
      using sub by blast
    then have xSigma:x∈⋃T×⋃S using Top_1_4_T1(3) assms(1,2) by auto
    then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S using Pair_fst_snd_eq by auto
    then have xT:fst(x)∈⋃T and xS:snd(x)∈⋃S by auto
    {
      fix U V assume as:U∈T  fst(x)∈U
      have ⋃S∈S using assms(2) unfolding IsATopology_def by auto
      with as have U×(⋃S)∈ProductCollection(T,S) unfolding ProductCollection_def
        by auto
      then have P:U×(⋃S)∈ProductTopology(T,S) using Top_1_4_T1(2) assms(1,2)
base_sets_open by blast
      with xS as(2) have ⟨fst(x),snd(x)⟩∈U×(⋃S) by auto
      then have x∈U×(⋃S) using Pair_fst_snd_eq xSigma by auto
      with P reg have U×(⋃S)∩A×B≠0 by auto
      then have noEm:U∩A≠0 by auto
    }
    then have ∀U∈T. fst(x)∈U ⟶ U∩A≠0 by auto moreover
    {
      fix U V assume as:U∈S  snd(x)∈U
      have ⋃T∈T using assms(1) unfolding IsATopology_def by auto
      with as have (⋃T)×U∈ProductCollection(T,S) unfolding ProductCollection_def
        by auto
      then have P:(⋃T)×U∈ProductTopology(T,S) using Top_1_4_T1(2) assms(1,2)
base_sets_open by blast
      with xT as(2) have ⟨fst(x),snd(x)⟩∈(⋃T)×U by auto
      then have x∈(⋃T)×U using Pair_fst_snd_eq xSigma by auto
      with P reg have (⋃T)×U∩A×B≠0 by auto
```

```
      then have noEm:U∩B≠0 by auto
    }
    then have ∀U∈S. snd(x)∈U ⟶ U∩B≠0 by auto
    ultimately have fst(x)∈Closure(A,T) snd(x)∈Closure(B,S) using
      topology0.inter_neigh_cl assms(3,4) unfolding topology0_def
      using assms(1,2) xT xS by auto
    then have ⟨fst(x),snd(x)⟩∈Closure(A,T)×Closure(B,S) by auto
    with xSigma have x∈Closure(A,T)×Closure(B,S) by auto
  }
  then show Closure(A×B,ProductTopology(T,S))⊆Closure(A,T)×Closure(B,S)
by auto
  {
    fix x assume x:x∈Closure(A,T)×Closure(B,S)
    then have xcl:fst(x)∈Closure(A,T) snd(x)∈Closure(B,S) by auto
    from xcl(1) have regT:∀U∈T. fst(x)∈U ⟶ U∩A≠0 using topology0.cl_inter_neigh
      unfolding topology0_def using assms(1,3) by blast
    from xcl(2) have regS:∀U∈S. snd(x)∈U ⟶ U∩B≠0 using topology0.cl_inter_neigh
      unfolding topology0_def using assms(2,4) by blast
    from x assms(3,4) have x∈⋃T×⋃S using topology0.Top_3_L11(1) un-
folding topology0_def
      using assms(1,2) by blast
    then have xtot:x∈⋃ProductTopology(T,S) using Top_1_4_T1(3) assms(1,2)
by auto
    {
      fix PO assume as:PO∈ProductTopology(T,S) x∈PO
      then obtain POB where base:POB∈ProductCollection(T,S) x∈POBPOB⊆PO
using point_open_base_neigh
        Top_1_4_T1(2) assms(1,2) base_sets_open by blast
      then obtain VT VS where V:VT∈T VS∈S x∈VT×VS POB=VT×VS unfold-
ing ProductCollection_def
        by auto
      from V(3) have x:fst(x)∈VT snd(x)∈VS by auto
      from V(1) regT x(1) have VT∩A≠0 by auto moreover
      from V(2) regS x(2) have VS∩B≠0 by auto ultimately
      have VT×VS∩A×B≠0 by auto
      with V(4) base(3) have PO∩A×B≠0 by blast
    }
    then have ∀P∈ProductTopology(T,S). x∈P ⟶ P∩A×B≠0 by auto
    then have x∈Closure(A×B,ProductTopology(T,S)) using topology0.inter_neigh_cl
      unfolding topology0_def using top sub xtot by auto
  }
  then show Closure(A,T)×Closure(B,S)⊆Closure(A×B,ProductTopology(T,S))
by auto
qed
```

The product of closed sets, is closed in the product topology.

```
corollary closed_product:
  assumes T{is a topology} S{is a topology} A{is closed in}TB{is closed
in}S
```

```
  shows (A×B) {is closed in}ProductTopology(T,S)
proof-
  from assms(3,4) have sub:A⊆⋃TB⊆⋃S unfolding IsClosed_def by auto
  then have A×B⊆⋃T×⋃S by auto
  then have sub1:A×B⊆⋃ProductTopology(T,S) using Top_1_4_T1(3) assms(1,2)
by auto
  from sub assms have Closure(A,T)=AClosure(B,S)=B using topology0.Top_3_L8
    unfolding topology0_def by auto
  then have Closure(A×B,ProductTopology(T,S))=A×B using cl_product
    assms(1,2) sub by auto
  then show thesis using topology0.Top_3_L8 unfolding topology0_def
    using sub1 Top_1_4_T1(1) assms(1,2) by auto
qed
```

86.2 Separation properties in product space

The product of T_0 spaces is T_0.

```
theorem T0_product:
  assumes T{is a topology}S{is a topology}T{is T0}S{is T0}
  shows ProductTopology(T,S){is T0}
proof-
  {
    fix x y assume x∈⋃ProductTopology(T,S)y∈⋃ProductTopology(T,S)x≠y
    then have tot:x∈⋃T×⋃Sy∈⋃T×⋃Sx≠y using Top_1_4_T1(3) assms(1,2)
by auto
    then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S⟨fst(y),snd(y)⟩∈⋃T×⋃S and disj:fst(x)≠fst(y)∨snd(x

      using Pair_fst_snd_eq by auto
    then have T:fst(x)∈⋃Tfst(y)∈⋃T and S:snd(y)∈⋃Ssnd(x)∈⋃S and
p:fst(x)≠fst(y)∨snd(x)≠snd(y)
      by auto
    {
      assume fst(x)≠fst(y)
      with T assms(3) have (∃U∈T. (fst(x)∈U∧fst(y)∉U)∨(fst(y)∈U∧fst(x)∉U))
unfolding
        isT0_def by auto
      then obtain U where U∈T (fst(x)∈U∧fst(y)∉U)∨(fst(y)∈U∧fst(x)∉U)
by auto
      with S have (⟨fst(x),snd(x)⟩∈U×(⋃S) ∧ ⟨fst(y),snd(y)⟩∉U×(⋃S))∨(⟨fst(y),snd(y)⟩∈U×(
∧ ⟨fst(x),snd(x)⟩∉U×(⋃S))
        by auto
      then have (x∈U×(⋃S) ∧ y∉U×(⋃S))∨(y∈U×(⋃S) ∧ x∉U×(⋃S)) us-
ing Pair_fst_snd_eq tot(1,2) by auto
      moreover have (⋃S)∈S using assms(2) unfolding IsATopology_def
by auto
      with ‹U∈T› have U×(⋃S)∈ProductTopology(T,S) using prod_open_open_prod
assms(1,2) by auto
      ultimately
      have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V)∨(y∈V ∧ x∉V) proof qed
```

```
    } moreover
    {
      assume snd(x)≠snd(y)
      with S assms(4) have (∃U∈S. (snd(x)∈U∧snd(y)∉U)∨(snd(y)∈U∧snd(x)∉U))
unfolding
        isT0_def by auto
      then obtain U where U∈S (snd(x)∈U∧snd(y)∉U)∨(snd(y)∈U∧snd(x)∉U)
by auto
      with T have (⟨fst(x),snd(x)⟩∈(⋃T)×U ∧ ⟨fst(y),snd(y)⟩∉(⋃T)×U)∨(⟨fst(y),snd(y)⟩∈(⋃
∧ ⟨fst(x),snd(x)⟩∉(⋃T)×U)
        by auto
      then have (x∈(⋃T)×U ∧ y∉(⋃T)×U)∨(y∈(⋃T)×U ∧ x∉(⋃T)×U) us-
ing Pair_fst_snd_eq tot(1,2) by auto
      moreover have (⋃T)∈T using assms(1) unfolding IsATopology_def
by auto
      with ‹U∈S› have (⋃T)×U∈ProductTopology(T,S) using prod_open_open_prod
assms(1,2) by auto
      ultimately
      have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V)∨(y∈V ∧ x∉V) proof qed
    }moreover
    note disj
    ultimately have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V)∨(y∈V ∧ x∉V)
by auto
  }
  then show thesis unfolding isT0_def by auto
qed
```

The product of T_1 spaces is T_1.

```
theorem T1_product:
  assumes T{is a topology}S{is a topology}T{is T₁}S{is T₁}
  shows ProductTopology(T,S){is T₁}
proof-
  {
    fix x y assume x∈⋃ProductTopology(T,S)y∈⋃ProductTopology(T,S)x≠y
    then have tot:x∈⋃T×⋃Sy∈⋃T×⋃Sx≠y using Top_1_4_T1(3) assms(1,2)
by auto
    then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S⟨fst(y),snd(y)⟩∈⋃T×⋃S and disj:fst(x)≠fst(y)∨snd(x

      using Pair_fst_snd_eq by auto
    then have T:fst(x)∈⋃Tfst(y)∈⋃T and S:snd(y)∈⋃Ssnd(x)∈⋃S and
p:fst(x)≠fst(y)∨snd(x)≠snd(y)
      by auto
    {
      assume fst(x)≠fst(y)
      with T assms(3) have (∃U∈T. (fst(x)∈U∧fst(y)∉U)) unfolding
        isT1_def by auto
      then obtain U where U∈T (fst(x)∈U∧fst(y)∉U) by auto
      with S have (⟨fst(x),snd(x)⟩∈U×(⋃S) ∧ ⟨fst(y),snd(y)⟩∉U×(⋃S))
by auto
```

```
      then have (x∈U×(⋃S) ∧ y∉U×(⋃S)) using Pair_fst_snd_eq tot(1,2)
by auto
      moreover have (⋃S)∈S using assms(2) unfolding IsATopology_def
by auto
      with ‹U∈T› have U×(⋃S)∈ProductTopology(T,S) using prod_open_open_prod
assms(1,2) by auto
      ultimately
      have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V) proof qed
    } moreover
    {
      assume snd(x)≠snd(y)
      with S assms(4) have (∃U∈S. (snd(x)∈U∧snd(y)∉U)) unfolding
        isT1_def by auto
      then obtain U where U∈S (snd(x)∈U∧snd(y)∉U) by auto
      with T have (⟨fst(x),snd(x)⟩∈(⋃T)×U ∧ ⟨fst(y),snd(y)⟩∉(⋃T)×U)
by auto
      then have (x∈(⋃T)×U ∧ y∉(⋃T)×U) using Pair_fst_snd_eq tot(1,2)
by auto
      moreover have (⋃T)∈T using assms(1) unfolding IsATopology_def
by auto
      with ‹U∈S› have (⋃T)×U∈ProductTopology(T,S) using prod_open_open_prod
assms(1,2) by auto
      ultimately
      have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V) proof qed
    }moreover
    note disj
    ultimately have ∃V∈ProductTopology(T,S). (x∈V ∧ y∉V) by auto
  }
  then show thesis unfolding isT1_def by auto
qed
```

The product of T_2 spaces is T_2.

```
theorem T2_product:
  assumes T{is a topology}S{is a topology}T{is T2}S{is T2}
  shows ProductTopology(T,S){is T2}
proof-
  {
    fix x y assume x∈⋃ProductTopology(T,S)y∈⋃ProductTopology(T,S)x≠y
    then have tot:x∈⋃T×⋃Sy∈⋃T×⋃Sx≠y using Top_1_4_T1(3) assms(1,2)
by auto
    then have ⟨fst(x),snd(x)⟩∈⋃T×⋃S⟨fst(y),snd(y)⟩∈⋃T×⋃S and disj:fst(x)≠fst(y)∨snd(x

      using Pair_fst_snd_eq by auto
    then have T:fst(x)∈⋃Tfst(y)∈⋃T and S:snd(y)∈⋃Ssnd(x)∈⋃S and
p:fst(x)≠fst(y)∨snd(x)≠snd(y)
      by auto
    {
      assume fst(x)≠fst(y)
      with T assms(3) have (∃U∈T. ∃V∈T. (fst(x)∈U∧fst(y)∈V) ∧ U∩V=0)
```

```
unfolding
        isT2_def by auto
      then obtain U V where U∈T V∈T fst(x)∈U fst(y)∈V U∩V=0 by auto
      with S have ⟨fst(x),snd(x)⟩∈U×(⋃S) ⟨fst(y),snd(y)⟩∈V×(⋃S) and
disjoint:(U×⋃S)∩(V×⋃S)=0 by auto
      then have x∈U×(⋃S)y∈V×(⋃S) using Pair_fst_snd_eq tot(1,2) by
auto
      moreover have (⋃S)∈S using assms(2) unfolding IsATopology_def
by auto
      with ‹U∈T›‹V∈T› have P:U×(⋃S)∈ProductTopology(T,S) V×(⋃S)∈ProductTopology(T,S)

        using prod_open_open_prod assms(1,2) by auto
      note disjoint ultimately
      have x∈U×(⋃S) ∧ y∈V×(⋃S) ∧ (U×(⋃S))∩(V×(⋃S))=0 by auto
      with P(2) have ∃UU∈ProductTopology(T,S). (x∈U×(⋃S) ∧ y∈UU ∧
(U×(⋃S))∩UU=0)
        using exI[where x=V×(⋃S) and P=λt. t∈ProductTopology(T,S) ∧
(x∈U×(⋃S) ∧ y∈t ∧ (U×(⋃S))∩t=0)] by auto
      with P(1) have ∃VV∈ProductTopology(T,S). ∃UU∈ProductTopology(T,S).
(x∈VV ∧ y∈UU ∧ VV∩UU=0)
        using exI[where x=U×(⋃S) and P=λt. t∈ProductTopology(T,S) ∧
(∃UU∈ProductTopology(T,S). (x∈t ∧ y∈UU ∧ (t)∩UU=0))] by auto
    } moreover
    {
      assume snd(x)≠snd(y)
      with S assms(4) have (∃U∈S. ∃V∈S. (snd(x)∈U∧snd(y)∈V) ∧ U∩V=0)
unfolding
        isT2_def by auto
      then obtain U V where U∈S V∈S snd(x)∈U snd(y)∈V U∩V=0 by auto
      with T have ⟨fst(x),snd(x)⟩∈(⋃T)×U ⟨fst(y),snd(y)⟩∈(⋃T)×V and
disjoint:((⋃T)×U)∩((⋃T)×V)=0 by auto
      then have x∈(⋃T)×Uy∈(⋃T)×V using Pair_fst_snd_eq tot(1,2) by
auto
      moreover have (⋃T)∈T using assms(1) unfolding IsATopology_def
by auto
      with ‹U∈S›‹V∈S› have P:(⋃T)×U∈ProductTopology(T,S) (⋃T)×V∈ProductTopology(T,S)

        using prod_open_open_prod assms(1,2) by auto
      note disjoint ultimately
      have x∈(⋃T)×U ∧ y∈(⋃T)×V ∧ ((⋃T)×U)∩((⋃T)×V)=0 by auto
      with P(2) have ∃UU∈ProductTopology(T,S). (x∈(⋃T)×U ∧ y∈UU ∧
((⋃T)×U)∩UU=0)
        using exI[where x=(⋃T)×V and P=λt. t∈ProductTopology(T,S) ∧
(x∈(⋃T)×U ∧ y∈t ∧ ((⋃T)×U)∩t=0)] by auto
      with P(1) have ∃VV∈ProductTopology(T,S). ∃UU∈ProductTopology(T,S).
(x∈VV ∧ y∈UU ∧ VV∩UU=0)
        using exI[where x=(⋃T)×U and P=λt. t∈ProductTopology(T,S) ∧
(∃UU∈ProductTopology(T,S). (x∈t ∧ y∈UU ∧ (t)∩UU=0))] by auto
    } moreover
```

```
    note disj
    ultimately have ∃VV∈ProductTopology(T, S). ∃UU∈ProductTopology(T,
S). x ∈ VV ∧ y ∈ UU ∧ VV ∩ UU = 0 by auto
  }
  then show thesis unfolding isT2_def by auto
qed
```

The product of regular spaces is regular.

```
theorem regular_product:
  assumes T{is a topology} S{is a topology} T{is regular} S{is regular}
  shows ProductTopology(T,S){is regular}
proof-
  {
    fix x U assume x∈⋃ProductTopology(T,S) U∈ProductTopology(T,S) x∈U
    then obtain V W where VW:V∈TW∈S V×W⊆U and x:x∈V×W using prod_top_point_neighb

      assms(1,2) by blast
    then have p:fst(x)∈Vsnd(x)∈W by auto
    from p(1) ‹V∈T› obtain VV where VV:fst(x)∈VV Closure(VV,T)⊆V VV∈T
using
      assms(1,3) topology0.regular_imp_exist_clos_neig unfolding topology0_def
      by force moreover
    from p(2) ‹W∈S› obtain WW where WW:snd(x)∈WW Closure(WW,S)⊆W WW∈S
using
      assms(2,4) topology0.regular_imp_exist_clos_neig unfolding topology0_def
      by force ultimately
    have  x∈VV×WW using x by auto
    moreover from  ‹Closure(VV,T)⊆V› ‹Closure(WW,S)⊆W› have Closure(VV,T)×Closure(WW,S)
⊆ V×W
      by auto
    moreover from VV(3) WW(3) have VV⊆⋃TWW⊆⋃S by auto
    ultimately have x∈VV×WW Closure(VV×WW,ProductTopology(T,S)) ⊆ V×W
using cl_product assms(1,2)
      by auto
    moreover have VV×WW∈ProductTopology(T,S) using prod_open_open_prod
assms(1,2)
      VV(3) WW(3) by auto
    ultimately have ∃Z∈ProductTopology(T,S). x∈Z ∧ Closure(Z,ProductTopology(T,S))⊆V×W
by auto
    with VW(3) have ∃Z∈ProductTopology(T,S). x∈Z ∧ Closure(Z,ProductTopology(T,S))⊆U
by auto
  }
  then have ∀x∈⋃ProductTopology(T,S). ∀U∈ProductTopology(T,S).x∈U ⟶
(∃Z∈ProductTopology(T,S). x∈Z ∧ Closure(Z,ProductTopology(T,S))⊆U)
    by auto
  then show thesis using topology0.exist_clos_neig_imp_regular unfold-
ing topology0_def
    using assms(1,2) Top_1_4_T1(1) by auto
qed
```

86.3 Connection properties in product space

First, we prove that the projection functions are open.

```
lemma projection_open:
  assumes T{is a topology}S{is a topology}B∈ProductTopology(T,S)
  shows {y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B}∈T
proof-
  {
    fix z assume z∈{y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B}
    then obtain x where x:x∈⋃S and z:z∈⋃T and p:⟨z,x⟩∈B by auto
    then have z∈{y∈⋃T. ⟨y,x⟩∈B} {y∈⋃T. ⟨y,x⟩∈B}⊆{y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B}
by auto moreover
    from x have {y∈⋃T. ⟨y,x⟩∈B}∈T using prod_sec_open2 assms by auto
    ultimately have ∃V∈T. z∈V ∧ V⊆{y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B} unfolding
Bex_def by auto
  }
  then show {y∈⋃T. ∃x∈⋃S. ⟨y,x⟩∈B}∈T using topology0.open_neigh_open
unfolding topology0_def
    using assms(1) by blast
qed

lemma projection_open2:
  assumes T{is a topology}S{is a topology}B∈ProductTopology(T,S)
  shows {y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B}∈S
proof-
  {
    fix z assume z∈{y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B}
    then obtain x where x:x∈⋃T and z:z∈⋃S and p:⟨x,z⟩∈B by auto
    then have z∈{y∈⋃S. ⟨x,y⟩∈B} {y∈⋃S. ⟨x,y⟩∈B}⊆{y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B}
by auto moreover
    from x have {y∈⋃S. ⟨x,y⟩∈B}∈S using prod_sec_open1 assms by auto
    ultimately have ∃V∈S. z∈V ∧ V⊆{y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B} unfolding
Bex_def by auto
  }
  then show {y∈⋃S. ∃x∈⋃T. ⟨x,y⟩∈B}∈S using topology0.open_neigh_open
unfolding topology0_def
    using assms(2) by blast
qed
```

The product of connected spaces is connected.

```
theorem compact_product:
  assumes T{is a topology}S{is a topology}T{is connected}S{is connected}
  shows ProductTopology(T,S){is connected}
proof-
  {
    fix U assume U:U∈ProductTopology(T,S) U{is closed in}ProductTopology(T,S)
    then have P:U∈ProductTopology(T,S) ⋃ProductTopology(T,S)-U∈ProductTopology(T,S)
      unfolding IsClosed_def by auto
    {
```

```
      fix s assume s:s∈⋃S
      with P(1) have p:{x∈⋃T. ⟨x,s⟩∈U}∈T using prod_sec_open2 assms(1,2)
by auto
      from s P(2) have oop:{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)}∈T
using prod_sec_open2
        assms(1,2) by blast
      then have ⋃T-(⋃T-{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)})={y∈⋃T.
⟨y,s⟩∈(⋃ProductTopology(T,S)-U)} by auto
      with oop have cl:(⋃T-{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)})
{is closed in}T unfolding IsClosed_def by auto
      {
        fix t assume t∈⋃T-{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)}
        then have tt:t∈⋃T t∉{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)}
by auto
        then have ⟨t,s⟩∉(⋃ProductTopology(T,S)-U) by auto
        then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃ProductTopology(T,S) by auto
        then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃T×⋃S using Top_1_4_T1(3) assms(1,2)
by auto
        with tt(1) s have ⟨t,s⟩∈U by auto
        with tt(1) have t∈{x∈⋃T. ⟨x,s⟩∈U} by auto
      } moreover
      {
        fix t assume t∈{x∈⋃T. ⟨x,s⟩∈U}
        then have tt:t∈⋃T ⟨t,s⟩∈U by auto
        then have ⟨t,s⟩∉⋃ProductTopology(T,S)-U by auto
        then have t∉{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)} by auto
        with tt(1) have t∈⋃T-{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)}
by auto
      }
      ultimately have {x∈⋃T. ⟨x,s⟩∈U}=⋃T-{y∈⋃T. ⟨y,s⟩∈(⋃ProductTopology(T,S)-U)}
by blast
      with cl have {x∈⋃T. ⟨x,s⟩∈U}{is closed in}T by auto
      with p assms(3) have {x∈⋃T. ⟨x,s⟩∈U}=0 ∨ {x∈⋃T. ⟨x,s⟩∈U}=⋃T
unfolding IsConnected_def
        by auto moreover
      {
        assume {x∈⋃T. ⟨x,s⟩∈U}=0
        then have ∀x∈⋃T. ⟨x,s⟩∉U by auto
      }
      moreover
      {
        assume AA:{x∈⋃T. ⟨x,s⟩∈U}=⋃T
        {
          fix x assume x∈⋃T
          with AA have x∈{x∈⋃T. ⟨x,s⟩∈U} by auto
          then have ⟨x,s⟩∈U by auto
        }
        then have ∀x∈⋃T. ⟨x,s⟩∈U by auto
      }
```

```
      ultimately have (∀x∈⋃T. ⟨x,s⟩∉U) ∨ (∀x∈⋃T. ⟨x,s⟩∈U) by blast
    }
    then have reg:∀s∈⋃S. (∀x∈⋃T. ⟨x,s⟩∉U) ∨ (∀x∈⋃T. ⟨x,s⟩∈U) by auto
    {
      fix q assume qU:q∈⋃T×{snd(qq). qq∈U}
      then obtain t u where t:t∈⋃T u∈U q=⟨t,snd(u)⟩ by auto
      with U(1) have u∈⋃ProductTopology(T,S) by auto
      then have u∈⋃T×⋃S using Top_1_4_T1(3) assms(1,2) by auto more-
over
      then have uu:u=⟨fst(u),snd(u)⟩ using Pair_fst_snd_eq by auto ul-
timately
      have fu:fst(u)∈⋃Tsnd(u)∈⋃S by (safe,auto)
      with reg have (∀tt∈⋃T. ⟨tt,snd(u)⟩∉U)∨(∀tt∈⋃T. ⟨tt,snd(u)⟩∈U)
by auto
      with ‹u∈U› uu fu(1) have ∀tt∈⋃T. ⟨tt,snd(u)⟩∈U by force
      with t(1,3) have q∈U by auto
    }
    then have U1:⋃T×{snd(qq). qq∈U}⊆U by auto
    {
      fix t assume t:t∈⋃T
      with P(1) have p:{x∈⋃S. ⟨t,x⟩∈U}∈S using prod_sec_open1 assms(1,2)
by auto
      from t P(2) have oop:{x∈⋃S. ⟨t,x⟩∈(⋃ProductTopology(T,S)-U)}∈S
using prod_sec_open1
        assms(1,2) by blast
      then have ⋃S-(⋃S-{x∈⋃S. ⟨t,x⟩∈(⋃ProductTopology(T,S)-U)})={y∈⋃S.
⟨t,y⟩∈(⋃ProductTopology(T,S)-U)} by auto
      with oop have cl:(⋃S-{y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)})
{is closed in}S unfolding IsClosed_def by auto
      {
        fix s assume s∈⋃S-{y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)}
        then have tt:s∈⋃S s∉{y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)}
by auto
        then have ⟨t,s⟩∉(⋃ProductTopology(T,S)-U) by auto
        then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃ProductTopology(T,S) by auto
        then have ⟨t,s⟩∈U ∨ ⟨t,s⟩∉⋃T×⋃S using Top_1_4_T1(3) assms(1,2)
by auto
        with tt(1) t have ⟨t,s⟩∈U by auto
        with tt(1) have s∈{x∈⋃S. ⟨t,x⟩∈U} by auto
      } moreover
      {
        fix s assume s∈{x∈⋃S. ⟨t,x⟩∈U}
        then have tt:s∈⋃S ⟨t,s⟩∈U by auto
        then have ⟨t,s⟩∉⋃ProductTopology(T,S)-U by auto
        then have s∉{y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)} by auto
        with tt(1) have s∈⋃S-{y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)}
by auto
      }
      ultimately have {x∈⋃S. ⟨t,x⟩∈U}=⋃S-{y∈⋃S. ⟨t,y⟩∈(⋃ProductTopology(T,S)-U)}
```

```
by blast
      with cl have {x∈⋃S. ⟨t,x⟩∈U}{is closed in}S by auto
      with p assms(4) have {x∈⋃S. ⟨t,x⟩∈U}=0 ∨ {x∈⋃S. ⟨t,x⟩∈U}=⋃S
unfolding IsConnected_def
        by auto moreover
      {
        assume {x∈⋃S. ⟨t,x⟩∈U}=0
        then have ∀x∈⋃S. ⟨t,x⟩∉U by auto
      }
      moreover
      {
        assume AA:{x∈⋃S. ⟨t,x⟩∈U}=⋃S
        {
          fix x assume x∈⋃S
          with AA have x∈{x∈⋃S. ⟨t,x⟩∈U} by auto
          then have ⟨t,x⟩∈U by auto
        }
        then have ∀x∈⋃S. ⟨t,x⟩∈U by auto
      }
      ultimately have (∀x∈⋃S. ⟨t,x⟩∉U) ∨ (∀x∈⋃S. ⟨t,x⟩∈U) by blast
    }
    then have reg:∀s∈⋃T. (∀x∈⋃S. ⟨s,x⟩∉U) ∨ (∀x∈⋃S. ⟨s,x⟩∈U) by auto
    {
      fix q assume qU:q∈{fst(qq). qq∈U}×⋃S
      then obtain qq s where t:q=⟨fst(qq),s⟩ qq∈U s∈⋃S by auto
      with U(1) have qq∈⋃ProductTopology(T,S) by auto
      then have qq∈⋃T×⋃S using Top_1_4_T1(3) assms(1,2) by auto more-
over
      then have qq:qq=⟨fst(qq),snd(qq)⟩ using Pair_fst_snd_eq by auto
ultimately
      have fq:fst(qq)∈⋃Tsnd(qq)∈⋃S by (safe,auto)
      from fq(1) reg have (∀tt∈⋃S. ⟨fst(qq),tt⟩∉U)∨(∀tt∈⋃S. ⟨fst(qq),tt⟩∈U)
by auto moreover
      with ‹qq∈U› qq fq(2) have ∀tt∈⋃S. ⟨fst(qq),tt⟩∈U by force
      with t(1,3) have q∈U by auto
    }
    then have U2:{fst(qq). qq∈U}×⋃S⊆U by blast
    {
      assume U≠0
      then obtain u where u:u∈U by auto
      {
        fix aa assume aa∈⋃T×⋃S
        then obtain t s where t∈⋃Ts∈⋃Saa=⟨t,s⟩ by auto
        with u have ⟨t,snd(u)⟩∈⋃T×{snd(qq). qq∈U} by auto
        with U1 have ⟨t,snd(u)⟩∈U by auto
        moreover have t=fst(⟨t,snd(u)⟩) by auto moreover note ‹s∈⋃S›
ultimately
        have ⟨t,s⟩∈{fst(qq). qq∈U}×⋃S by blast
        with U2 have ⟨t,s⟩∈U by auto
```

```
        with ‹aa=⟨t,s⟩› have aa∈U by auto
      }
      then have ⋃T×⋃S⊆U by auto moreover
      with U(1) have U⊆⋃ProductTopology(T,S) by auto ultimately
      have ⋃T×⋃S=U using Top_1_4_T1(3) assms(1,2) by auto
    }
    then have (U=0)∨(U=⋃T×⋃S) by auto
  }
  then show thesis unfolding IsConnected_def using Top_1_4_T1(3) assms(1,2)
by auto
qed

end
```

87 Topology 11

```
theory Topology_ZF_11 imports Topology_ZF_7 Finite_ZF_1

begin
```

This file deals with order topologies. The order topology is already defined in `Topology_ZF_examples_1.thy`.

87.1 Order topologies

We will assume most of the time that the ordered set has more than one point. It is natural to think that the topological properties can be translated to properties of the order; since every order rises one and only one topology in a set.

87.2 Separation properties

Order topologies have a lot of separation properties.

Every order topology is Hausdorff.

```
theorem order_top_T2:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X
  shows (OrdTopology X r){is T₂}
proof-
  {
    fix x y assume A1:x∈⋃(OrdTopology X r)y∈⋃(OrdTopology X r)x≠y
    then have AS:x∈Xy∈Xx≠y using union_ordtopology[OF assms(1) assms(2)]
by auto
    {
      assume A2:∃z∈X-{x,y}. (⟨x,y⟩∈r⟶⟨x,z⟩∈r∧⟨z,y⟩∈r)∧(⟨y,x⟩∈r⟶⟨y,z⟩∈r∧⟨z,x⟩∈r)
      from AS(1,2) assms(1) have ⟨x,y⟩∈r∨⟨y,x⟩∈r unfolding IsLinOrder_def
IsTotal_def by auto moreover
```

```
      {
        assume ⟨x,y⟩∈r
        with AS A2 obtain z where z:⟨x,z⟩∈r⟨z,y⟩∈rz∈Xz≠xz≠y by auto
        with AS(1,2) have x∈LeftRayX(X,r,z)y∈RightRayX(X,r,z) unfold-
ing LeftRayX_def RightRayX_def
          by auto moreover
        have LeftRayX(X,r,z)∩RightRayX(X,r,z)=0 using inter_lray_rray[OF
z(3) z(3) assms(1)]
          unfolding IntervalX_def using Order_ZF_2_L4[OF total_is_refl
_ z(3)] assms(1) unfolding IsLinOrder_def
          by auto moreover
        have LeftRayX(X,r,z)∈(OrdTopology X r)RightRayX(X,r,z)∈(OrdTopology
X r)
          using z(3) base_sets_open[OF Ordtopology_is_a_topology(2)[OF
assms(1)]] by auto
        ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
      }
      moreover
      {
        assume ⟨y,x⟩∈r
        with AS A2 obtain z where z:⟨y,z⟩∈r⟨z,x⟩∈rz∈Xz≠xz≠y by auto
        with AS(1,2) have y∈LeftRayX(X,r,z)x∈RightRayX(X,r,z) unfold-
ing LeftRayX_def RightRayX_def
          by auto moreover
        have LeftRayX(X,r,z)∩RightRayX(X,r,z)=0 using inter_lray_rray[OF
z(3) z(3) assms(1)]
          unfolding IntervalX_def using Order_ZF_2_L4[OF total_is_refl
_ z(3)] assms(1) unfolding IsLinOrder_def
          by auto moreover
        have LeftRayX(X,r,z)∈(OrdTopology X r)RightRayX(X,r,z)∈(OrdTopology
X r)
          using z(3) base_sets_open[OF Ordtopology_is_a_topology(2)[OF
assms(1)]] by auto
        ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
      }
      ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
    }
    moreover
    {
      assume A2:∀z∈X - {x, y}. (⟨x, y⟩ ∈ r ∧ (⟨x, z⟩ ∉ r ∨ ⟨z, y⟩ ∉ r))
∨ (⟨y, x⟩ ∈ r ∧ (⟨y, z⟩ ∉ r ∨ ⟨z, x⟩ ∉ r))
      from AS(1,2) assms(1) have disj:⟨x,y⟩∈r∨⟨y,x⟩∈r unfolding IsLinOrder_def
IsTotal_def by auto moreover
      {
        assume TT:⟨x,y⟩∈r
        with AS assms(1) have T:⟨y,x⟩∉r unfolding IsLinOrder_def antisym_def
```

```
by auto
        from TT AS(1-3) have x∈LeftRayX(X,r,y)y∈RightRayX(X,r,x) unfolding LeftRayX_def RightRayX_def
          by auto moreover
        {
          fix z assume z∈LeftRayX(X,r,y)∩RightRayX(X,r,x)
          then have ⟨z,y⟩∈r⟨x,z⟩∈rz∈X-{x,y} unfolding RightRayX_def LeftRayX_def
by auto
          with A2 T have False by auto
        }
        then have LeftRayX(X,r,y)∩RightRayX(X,r,x)=0 by auto moreover
        have LeftRayX(X,r,y)∈(OrdTopology X r)RightRayX(X,r,x)∈(OrdTopology
X r)
          using base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]]
AS by auto
        ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
      }
      moreover
      {
        assume TT:⟨y,x⟩∈r
        with AS assms(1) have T:⟨x,y⟩∉r unfolding IsLinOrder_def antisym_def
by auto
        from TT AS(1-3) have y∈LeftRayX(X,r,x)x∈RightRayX(X,r,y) unfolding LeftRayX_def RightRayX_def
          by auto moreover
        {
          fix z assume z∈LeftRayX(X,r,x)∩RightRayX(X,r,y)
          then have ⟨z,x⟩∈r⟨y,z⟩∈rz∈X-{x,y} unfolding RightRayX_def LeftRayX_def
by auto
          with A2 T have False by auto
        }
        then have LeftRayX(X,r,x)∩RightRayX(X,r,y)=0 by auto moreover
        have LeftRayX(X,r,x)∈(OrdTopology X r)RightRayX(X,r,y)∈(OrdTopology
X r)
          using base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]]
AS by auto
        ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
      }
      ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
    }
    ultimately have ∃U∈(OrdTopology X r). ∃V∈(OrdTopology X r). x∈U
∧ y∈V ∧ U∩V=0 by auto
  }
  then show thesis unfolding isT2_def by auto
qed
```

Every order topology is T_4, but the proof needs lots of machinery. At the end of the file, we will prove that every order topology is normal; sooner or later.

87.3 Connectedness properties

Connectedness is related to two properties of orders: completeness and density

Some order-dense properties:

```
definition
  IsDenseSub (_ {is dense in}_{with respect to}_) where
  A {is dense in}X{with respect to}r ≡
  ∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y  ⟶ (∃z∈A-{x,y}. ⟨x,z⟩∈r∧⟨z,y⟩∈r)

definition
  IsDenseUnp (_ {is not-properly dense in}_{with respect to}_) where
  A {is not-properly dense in}X{with respect to}r ≡
  ∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y  ⟶ (∃z∈A. ⟨x,z⟩∈r∧⟨z,y⟩∈r)

definition
  IsWeaklyDenseSub (_ {is weakly dense in}_{with respect to}_) where
  A {is weakly dense in}X{with respect to}r ≡
  ∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y  ⟶ ((∃z∈A-{x,y}. ⟨x,z⟩∈r∧⟨z,y⟩∈r)∨ IntervalX(X,r,x,y)=0)

definition
  IsDense (_ {is dense with respect to}_) where
  X {is dense with respect to}r ≡
  ∀x∈X. ∀y∈X. ⟨x,y⟩∈r ∧ x≠y  ⟶ (∃z∈X-{x,y}. ⟨x,z⟩∈r∧⟨z,y⟩∈r)

lemma dense_sub:
  shows (X {is dense with respect to}r) ⟷ (X {is dense in}X{with respect
to}r)
  unfolding IsDenseSub_def IsDense_def by auto

lemma not_prop_dense_sub:
  shows (A {is dense in}X{with respect to}r) ⟶ (A {is not-properly
dense in}X{with respect to}r)
  unfolding IsDenseSub_def IsDenseUnp_def by auto
```

In densely ordered sets, intervals are infinite.

```
theorem dense_order_inf_intervals:
  assumes IsLinOrder(X,r) IntervalX(X, r, b, c)≠0b∈Xc∈X X{is dense with
respect to}r
  shows ¬Finite(IntervalX(X, r, b, c))
proof
  assume fin:Finite(IntervalX(X, r, b, c))
  have sub:IntervalX(X, r, b, c)⊆X unfolding IntervalX_def by auto
```

```
  have p:Minimum(r,IntervalX(X, r, b, c))∈IntervalX(X, r, b, c) using
Finite_ZF_1_T2(2)[OF assms(1) Finite_Fin[OF fin sub] assms(2)]
    by auto
  then have ⟨b,Minimum(r,IntervalX(X, r, b, c))⟩∈rb≠Minimum(r,IntervalX(X,
r, b, c))
    unfolding IntervalX_def using Order_ZF_2_L1 by auto
  with assms(3,5) sub p obtain z1 where z1:z1∈Xz1≠bz1≠Minimum(r,IntervalX(X,
r, b, c))⟨b,z1⟩∈r⟨z1,Minimum(r,IntervalX(X, r, b, c))⟩∈r
    unfolding IsDense_def by blast
  from p have B:⟨Minimum(r,IntervalX(X, r, b, c)),c⟩∈r unfolding IntervalX_def
using Order_ZF_2_L1 by auto moreover
  have trans(r) using assms(1) unfolding IsLinOrder_def by auto more-
over
  note z1(5) ultimately have z1a:⟨z1,c⟩∈r unfolding trans_def by fast
  {
    assume z1=c
    with B have ⟨Minimum(r,IntervalX(X, r, b, c)),z1⟩∈r by auto
    with z1(5) have z1=Minimum(r,IntervalX(X, r, b, c)) using assms(1)
unfolding IsLinOrder_def antisym_def by auto
    then have False using z1(3) by auto
  }
  then have z1≠c by auto
  with z1(1,2,4) z1a have z1∈IntervalX(X, r, b, c) unfolding IntervalX_def
using Order_ZF_2_L1 by auto
  then have ⟨Minimum(r,IntervalX(X, r, b, c)),z1⟩∈r using Finite_ZF_1_T2(4)[OF
assms(1) Finite_Fin[OF fin sub] assms(2)] by auto
  with z1(5) have z1=Minimum(r,IntervalX(X, r, b, c)) using assms(1)
unfolding IsLinOrder_def antisym_def by auto
  with z1(3) show False by auto
qed
```

Left rays are infinite.

```
theorem dense_order_inf_lrays:
  assumes IsLinOrder(X,r) LeftRayX(X,r,c)≠0c∈X  X{is dense with respect
to}r
  shows ¬Finite(LeftRayX(X,r,c))
proof-
  from assms(2) obtain b where b∈X⟨b,c⟩∈rb≠c unfolding LeftRayX_def
by auto
  with assms(3) obtain z where z∈X-{b,c}⟨b,z⟩∈r⟨z,c⟩∈r using assms(4)
unfolding IsDense_def by auto
  then have IntervalX(X, r, b, c)≠0 unfolding IntervalX_def using Order_ZF_2_L1
by auto
  then have nFIN:¬Finite(IntervalX(X, r, b, c)) using dense_order_inf_intervals[OF
assms(1) _ _ assms(3,4)]
    ‹b∈X› by auto
  {
    fix d assume d∈IntervalX(X, r, b, c)
    then have ⟨b,d⟩∈r⟨d,c⟩∈rd∈Xd≠bd≠c unfolding IntervalX_def using Order_ZF_2_L1
```

```
by auto
    then have d∈LeftRayX(X,r,c) unfolding LeftRayX_def by auto
  }
  then have IntervalX(X, r, b, c)⊆LeftRayX(X,r,c) by auto
  with nFIN show thesis using subset_Finite by auto
qed
```

Right rays are infinite.

```
theorem dense_order_inf_rrays:
  assumes IsLinOrder(X,r) RightRayX(X,r,b)≠0b∈X  X{is dense with respect
to}r
  shows ¬Finite(RightRayX(X,r,b))
proof-
  from assms(2) obtain c where c∈X⟨b,c⟩∈rb≠c unfolding RightRayX_def
by auto
  with assms(3) obtain z where z∈X-{b,c}⟨b,z⟩∈r⟨z,c⟩∈r using assms(4)
unfolding IsDense_def by auto
  then have IntervalX(X, r, b, c)≠0 unfolding IntervalX_def using Order_ZF_2_L1
by auto
  then have nFIN:¬Finite(IntervalX(X, r, b, c)) using dense_order_inf_intervals[OF
assms(1) _ assms(3) _ assms(4)]
    ‹c∈X› by auto
  {
    fix d assume d∈IntervalX(X, r, b, c)
    then have ⟨b,d⟩∈r⟨d,c⟩∈rd∈Xd≠bd≠c unfolding IntervalX_def using Order_ZF_2_L1
by auto
    then have d∈RightRayX(X,r,b) unfolding RightRayX_def by auto
  }
  then have IntervalX(X, r, b, c)⊆RightRayX(X,r,b) by auto
  with nFIN show thesis using subset_Finite by auto
qed
```

The whole space in a densely ordered set is infinite.

```
corollary dense_order_infinite:
  assumes IsLinOrder(X,r)  X{is dense with respect to}r
    ∃x y. x≠y∧x∈X∧y∈X
  shows ¬(X≺nat)
proof-
  from assms(3) obtain b c where B:b∈Xc∈Xb≠c by auto
  {
    assume ⟨b,c⟩∉r
    with assms(1) have ⟨c,b⟩∈r unfolding IsLinOrder_def IsTotal_def us-
ing ‹b∈X›‹c∈X› by auto
    with assms(2) B obtain z where z∈X-{b,c}⟨c,z⟩∈r⟨z,b⟩∈r unfolding
IsDense_def by auto
    then have IntervalX(X,r,c,b)≠0 unfolding IntervalX_def using Order_ZF_2_L1
by auto
    then have ¬(Finite(IntervalX(X,r,c,b))) using dense_order_inf_intervals[OF
assms(1) _ ‹c∈X›‹b∈X› assms(2)]
```

```
      by auto moreover
    have IntervalX(X,r,c,b)⊆X unfolding IntervalX_def by auto
    ultimately have ¬(Finite(X)) using subset_Finite by auto
    then have ¬(X≺nat) using lesspoll_nat_is_Finite by auto
  }
  moreover
  {
    assume ⟨b,c⟩∈r
    with assms(2) B obtain z where z∈X-{b,c}⟨b,z⟩∈r⟨z,c⟩∈r unfolding
IsDense_def by auto
    then have IntervalX(X,r,b,c)≠0 unfolding IntervalX_def using Order_ZF_2_L1
by auto
    then have ¬(Finite(IntervalX(X,r,b,c))) using dense_order_inf_intervals[OF
assms(1) _ ‹b∈X›‹c∈X› assms(2)]
      by auto moreover
    have IntervalX(X,r,b,c)⊆X unfolding IntervalX_def by auto
    ultimately have ¬(Finite(X)) using subset_Finite by auto
    then have ¬(X≺nat) using lesspoll_nat_is_Finite by auto
  }
  ultimately show thesis by auto
qed
```

If an order topology is connected, then the order is complete. It is equivalent to assume that $r \subseteq X \times X$ or prove that $r \cap X \times X$ is complete.

```
theorem conn_imp_complete:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X r⊆X×X
     (OrdTopology X r){is connected}
  shows r{is complete}
proof-
  {
    assume ¬(r{is complete})
    then obtain A where A:A≠0IsBoundedAbove(A,r)¬(HasAminimum(r, ⋂b∈A.
r  {b})) unfolding
      IsComplete_def by auto
    from A(3) have r1:∀m∈⋂b∈A. r  {b}. ∃x∈⋂b∈A. r  {b}. ⟨m,x⟩∉r un-
folding HasAminimum_def
      by force
    from A(1,2) obtain b where r2:∀x∈A. ⟨x, b⟩ ∈ r unfolding IsBoundedAbove_def
by auto
    with assms(3) A(1) have A⊆Xb∈X by auto
    with assms(3) have r3:∀c∈A. r  {c}⊆X using image_iff by auto
    from r2 have ∀x∈A. b∈r{x} using image_iff by auto
    then have noE:b∈(⋂b∈A. r  {b}) using A(1) by auto
    {
      fix x assume x∈(⋂b∈A. r  {b})
      then have ∀c∈A. x∈r{c} by auto
      with A(1) obtain c where c∈A x∈r{c} by auto
      with r3 have x∈X by auto
    }
```

```
    then have sub:(⋂b∈A. r  {b})⊆X by auto
    {
      fix x assume x:x∈(⋂b∈A. r  {b})
      with r1 have ∃z∈⋂b∈A. r  {b}. ⟨x,z⟩∉r by auto
      then obtain z where z:z∈(⋂b∈A. r  {b})⟨x,z⟩∉r by auto
      from x z(1) sub have x∈Xz∈X by auto
      with z(2) have ⟨z,x⟩∈r using assms(1) unfolding IsLinOrder_def IsTotal_def
by auto
      then have xx:x∈RightRayX(X,r,z) unfolding RightRayX_def using ‹x∈X›‹⟨x,z⟩∉r›
        assms(1) unfolding IsLinOrder_def using total_is_refl unfold-
ing refl_def by auto
      {
        fix m assume m∈RightRayX(X,r,z)
        then have m:m∈X-{z}⟨z,m⟩∈r unfolding RightRayX_def by auto
        {
          fix c assume c∈A
          with z(1) have ⟨c,z⟩∈r using image_iff by auto
          with m(2) have ⟨c,m⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast
          then have m∈r{c} using image_iff by auto
        }
        with A(1) have m∈(⋂b∈A. r  {b}) by auto
      }
      then have sub1:RightRayX(X,r,z)⊆(⋂b∈A. r  {b}) by auto
      have RightRayX(X,r,z)∈(OrdTopology X r) using
        base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]] ‹z∈X›
by auto
      with sub1 xx have ∃U∈(OrdTopology X r). x∈U ∧ U⊆(⋂b∈A. r  {b})
by auto
    }
    then have (⋂b∈A. r  {b})∈(OrdTopology X r) using topology0.open_neigh_open[OF
topology0_ordtopology[OF assms(1)]]
      by auto moreover
    {
      fix x assume x∈X-(⋂b∈A. r  {b})
      then have x∈Xx∉(⋂b∈A. r  {b}) by auto
      with A(1) obtain b where x∉r{b}b∈A by auto
      then have ⟨b,x⟩∉r using image_iff by auto
      with ‹A⊆X› ‹b∈A›‹x∈X› have ⟨x,b⟩∈r using assms(1) unfolding IsLinOrder_def
        IsTotal_def by auto
      then have xx:x∈LeftRayX(X,r,b) unfolding LeftRayX_def using ‹x∈X›
‹⟨b,x⟩∉r›
        assms(1) unfolding IsLinOrder_def using total_is_refl unfold-
ing refl_def by auto
      {
        fix y assume y∈LeftRayX(X,r,b)∩(⋂b∈A. r  {b})
        then have y∈X-{b}⟨y,b⟩∈r∀c∈A. y∈r{c} unfolding LeftRayX_def by
auto
        then have y∈X⟨y,b⟩∈r∀c∈A. ⟨c,y⟩∈r using image_iff by auto
```

```
        with ‹b∈A› have y=b using assms(1) unfolding IsLinOrder_def antisym_def
by auto
        then have False using ‹y∈X-{b}› by auto
      }
      then have sub1:LeftRayX(X,r,b)⊆X-(⋂b∈A. r  {b}) unfolding LeftRayX_def
by auto
      have LeftRayX(X,r,b)∈(OrdTopology X r) using
        base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]] ‹b∈A›‹A⊆X›
by blast
      with sub1 xx have ∃U∈(OrdTopology X r). x∈U∧U⊆X-(⋂b∈A. r  {b})
by auto
    }
    then have X - (⋂b∈A. r  {b})∈(OrdTopology X r) using topology0.open_neigh_open[OF
topology0_ordtopology[OF assms(1)]]
      by auto
    then have ⋃(OrdTopology X r)-(⋂b∈A. r  {b})∈(OrdTopology X r) us-
ing union_ordtopology[OF assms(1,2)] by auto
    then have (⋂b∈A. r  {b}){is closed in}(OrdTopology X r) unfolding
IsClosed_def using union_ordtopology[OF assms(1,2)]
      sub by auto
    moreover note assms(4) ultimately
    have (⋂b∈A. r  {b})=0∨(⋂b∈A. r  {b})=X using union_ordtopology[OF
assms(1,2)] unfolding IsConnected_def
      by auto
    then have e1:(⋂b∈A. r  {b})=X using noE by auto
    then have ∀x∈X. ∀b∈A. x∈r{b} by auto
    then have r4:∀x∈X. ∀b∈A. ⟨b,x⟩∈r using image_iff by auto
    {
      fix a1 a2 assume aA:a1∈Aa2∈Aa1≠a2
      with ‹A⊆X› have aX:a1∈Xa2∈X by auto
      with r4 aA(1,2) have ⟨a1,a2⟩∈r⟨a2,a1⟩∈r by auto
      then have a1=a2 using assms(1) unfolding IsLinOrder_def antisym_def
by auto
      with aA(3) have False by auto
    }
    moreover
    from A(1) obtain t where t∈A by auto
    ultimately have A={t} by auto
    with r4 have ∀x∈X. ⟨t,x⟩∈rt∈X using ‹A⊆X› by auto
    then have HasAminimum(r,X) unfolding HasAminimum_def by auto
    with e1 have HasAminimum(r,⋂b∈A. r  {b}) by auto
    with A(3) have False by auto
  }
  then show thesis by auto
qed
```

If an order topology is connected, then the order is dense.

```
theorem conn_imp_dense:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X
```

```
    (OrdTopology X r){is connected}
  shows X {is dense with respect to}r
proof-
  {
    assume ¬(X {is dense with respect to}r)
    then have ∃x1∈X. ∃x2∈X. ⟨x1,x2⟩∈r∧x1≠x2∧(∀z∈X-{x1,x2}. ⟨x1,z⟩∉r∨⟨z,x2⟩∉r)
      unfolding IsDense_def by auto
    then obtain x1 x2 where x:x1∈Xx2∈X⟨x1,x2⟩∈rx1≠x2(∀z∈X-{x1,x2}. ⟨x1,z⟩∉r∨⟨z,x2⟩∉r)
by auto
    from x(1,2) have P:LeftRayX(X,r,x2)∈(OrdTopology X r)RightRayX(X,r,x1)∈(OrdTopology
X r)
      using  base_sets_open[OF Ordtopology_is_a_topology(2)[OF assms(1)]]
by auto
    {
      fix x assume x∈X-LeftRayX(X,r,x2)
      then have x∈X x∉LeftRayX(X,r,x2) by auto
      then have ⟨x,x2⟩∉r∨x=x2 unfolding LeftRayX_def by auto
      then have ⟨x2,x⟩∈r∨x=x2 using assms(1) ‹x∈X› ‹x2∈X› unfolding IsLinOrder_def
        IsTotal_def by auto
      then have s:⟨x2,x⟩∈r using assms(1) unfolding IsLinOrder_def us-
ing total_is_refl ‹x2∈X›
        unfolding refl_def by auto
      with x(3) have ⟨x1,x⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast
      then have x=x1∨x∈RightRayX(X,r,x1) unfolding RightRayX_def us-
ing ‹x∈X› by auto
      with s have ⟨x2,x1⟩∈r∨x∈RightRayX(X,r,x1) by auto
      with x(3) have x1=x2 ∨ x∈RightRayX(X,r,x1) using assms(1) unfold-
ing IsLinOrder_def
        antisym_def by auto
      with x(4) have x∈RightRayX(X,r,x1) by auto
    }
    then have X-LeftRayX(X,r,x2)⊆RightRayX(X,r,x1) by auto moreover
    {
      fix x assume x∈RightRayX(X,r,x1)
      then have xr:x∈X-{x1}⟨x1,x⟩∈r unfolding RightRayX_def by auto
      {
        assume x∈LeftRayX(X,r,x2)
        then have xl:x∈X-{x2}⟨x,x2⟩∈r unfolding LeftRayX_def by auto
        from xl xr x(5) have False by auto
      }
      with xr(1) have x∈X-LeftRayX(X,r,x2) by auto
    }
    ultimately have RightRayX(X,r,x1)=X-LeftRayX(X,r,x2) by auto
    then have LeftRayX(X,r,x2){is closed in}(OrdTopology X r) using P(2)
union_ordtopology[
      OF assms(1,2)] unfolding IsClosed_def LeftRayX_def by auto
    with P(1) have LeftRayX(X,r,x2)=0∨LeftRayX(X,r,x2)=X using union_ordtopology[
      OF assms(1,2)] assms(3) unfolding IsConnected_def by auto
```

```
    with x(1,3,4) have LeftRayX(X,r,x2)=X unfolding LeftRayX_def by auto
    then have x2∈LeftRayX(X,r,x2) using x(2) by auto
    then have False unfolding LeftRayX_def by auto
  }
  then show thesis by auto
qed
```

Actually a connected order topology is one that comes from a dense and complete order.

First a lemma. In a complete ordered set, every non-empty set bounded from below has a maximum lower bound.

```
lemma complete_order_bounded_below:
  assumes r{is complete} IsBoundedBelow(A,r) A≠0 r⊆X×X
  shows HasAmaximum(r,⋂c∈A. r-{c})
proof-
  let M=⋂c∈A. r-{c}
  from assms(3) obtain t where A:t∈A by auto
  {
    fix m assume m∈M
    with A have m∈r-{t} by auto
    then have ⟨m,t⟩∈r by auto
  }
  then have (∀x∈⋂c∈A. r - {c}. ⟨x, t⟩ ∈ r) by auto
  then have IsBoundedAbove(M,r) unfolding IsBoundedAbove_def by auto
  moreover
  from assms(2,3) obtain l where  ∀x∈A. ⟨l, x⟩ ∈ r unfolding IsBoundedBelow_def
by auto
  then have ∀x∈A. l ∈ r-{x} using vimage_iff by auto
  with assms(3) have l∈M by auto
  then have M≠0 by auto moreover note assms(1)
  ultimately have HasAminimum(r,⋂c∈M. r  {c}) unfolding IsComplete_def
by auto
  then obtain rr where rr:rr∈(⋂c∈M. r  {c}) ∀s∈(⋂c∈M. r  {c}). ⟨rr,s⟩∈r
unfolding HasAminimum_def
    by auto
  {
    fix aa assume A:aa∈A
    {
      fix c assume M:c∈M
      with A have ⟨c,aa⟩∈r by auto
      then have aa∈r{c} by auto
    }
    then have aa∈(⋂c∈M. r  {c}) using rr(1) by auto
  }
  then have A⊆(⋂c∈M. r  {c}) by auto
  with rr(2) have ∀s∈A. ⟨rr,s⟩∈r by auto
  then have rr∈M using assms(3) by auto
  moreover
```

```
  {
    fix m assume m∈M
    then have rr∈r{m} using rr(1) by auto
    then have ⟨m,rr⟩∈r by auto
  }
  then have ∀m∈M. ⟨m,rr⟩∈r by auto
  ultimately show thesis unfolding HasAmaximum_def by auto
qed

theorem comp_dense_imp_conn:
  assumes IsLinOrder(X,r) ∃x y. x≠y∧x∈X∧y∈X r⊆X×X
     X {is dense with respect to}r r{is complete}
  shows (OrdTopology X r){is connected}
proof-
  {
    assume ¬((OrdTopology X r){is connected})
    then obtain U where U:U≠0U≠XU∈(OrdTopology X r)U{is closed in}(OrdTopology
X r)
      unfolding IsConnected_def using union_ordtopology[OF assms(1,2)]
by auto
    from U(4) have A:X-U∈(OrdTopology X r)U⊆X unfolding IsClosed_def
using union_ordtopology[OF assms(1,2)] by auto
    from U(1) obtain u where u∈U by auto
    from A(2) U(1,2) have X-U≠0 by auto
    then obtain v where v∈X-U by auto
    with ‹u∈U› ‹U⊆X› have ⟨u,v⟩∈r∨⟨v,u⟩∈r using assms(1) unfolding IsLinOrder_def
IsTotal_def
      by auto
    {
      assume ⟨u,v⟩∈r
      have LeftRayX(X,r,v)∈(OrdTopology X r) using base_sets_open[OF
Ordtopology_is_a_topology(2)[OF assms(1)]]
        ‹v∈X-U› by auto
      then have U∩LeftRayX(X,r,v)∈(OrdTopology X r) using U(3) using
Ordtopology_is_a_topology(1)
        [OF assms(1)] unfolding IsATopology_def by auto
      {
        fix b assume b∈(U)∩LeftRayX(X,r,v)
        then have ⟨b,v⟩∈r unfolding LeftRayX_def by auto
      }
      then have bound:IsBoundedAbove(U∩LeftRayX(X,r,v),r) unfolding IsBoundedAbove_def
by auto moreover
      with ‹⟨u,v⟩∈r›‹u∈U›‹U⊆X›‹v∈X-U› have nE:U∩LeftRayX(X,r,v)≠0 un-
folding LeftRayX_def by auto
      ultimately have Hmin:HasAminimum(r,⋂c∈U∩LeftRayX(X,r,v). r{c})
using assms(5) unfolding IsComplete_def
        by auto
      let min=Supremum(r,U∩LeftRayX(X,r,v))
      {
```

```
        fix c assume c∈U∩LeftRayX(X,r,v)
        then have ⟨c,v⟩∈r unfolding LeftRayX_def by auto
      }
      then have a1:⟨min,v⟩∈r using Order_ZF_5_L3[OF _ nE Hmin] assms(1)
unfolding IsLinOrder_def
        by auto
      {
        assume ass:min∈U
        then obtain V where V:min∈VV⊆U
          V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X} using point_open_base_neigh
          [OF Ordtopology_is_a_topology(2)[OF assms(1)] ‹U∈(OrdTopology
X r)› ass] by blast
        {
          assume V∈{RightRayX(X,r,b). b∈X}
          then obtain b where b:b∈X V=RightRayX(X,r,b) by auto
          note a1 moreover
          from V(1) b(2) have a2:⟨b,min⟩∈rmin≠b unfolding RightRayX_def
by auto
          ultimately have ⟨b,v⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by blast moreover
          {
            assume b=v
            with a1 a2(1) have b=min using assms(1) unfolding IsLinOrder_def
antisym_def by auto
            with a2(2) have False by auto
          }
          ultimately have False using V(2) b(2) unfolding RightRayX_def
using ‹v∈X-U› by auto
        }
        moreover
        {
          assume V∈{LeftRayX(X,r,b). b∈X}
          then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
          {
            assume ⟨v,b⟩∈r
            then have b=v∨v∈LeftRayX(X,r,b) unfolding LeftRayX_def us-
ing ‹v∈X-U› by auto
            then have b=v using b(1) V(2) ‹v∈X-U› by auto
          }
          then have bv:⟨b,v⟩∈r using assms(1) unfolding IsLinOrder_def
IsTotal_def using b(2)
            ‹v∈X-U› by auto
          from b(1) V(1) have ⟨min,b⟩∈rmin≠b unfolding LeftRayX_def by
auto
          with assms(4) obtain z where z:⟨min,z⟩∈r⟨z,b⟩∈rz∈X-{b,min}
unfolding IsDense_def
            using b(2) V(1,2) ‹U⊆X› by blast
          then have rayb:z∈LeftRayX(X,r,b) unfolding LeftRayX_def by
```

```
auto
          from z(2) bv have ⟨z,v⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast
          moreover
          {
            assume z=v
            with bv have ⟨b,z⟩∈r by auto
            with z(2) have b=z using assms(1) unfolding IsLinOrder_def
antisym_def by auto
            then have False using z(3) by auto
          }
          ultimately have z∈LeftRayX(X,r,v) unfolding LeftRayX_def us-
ing z(3) by auto
          with rayb have z∈U∩LeftRayX(X,r,v) using V(2) b(1) by auto
          then have min∈r{z} using Order_ZF_4_L4(1)[OF _ Hmin] assms(1)
unfolding Supremum_def IsLinOrder_def
            by auto
          then have ⟨z,min⟩∈r by auto
          with z(1,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
        }
        moreover
        {
          assume V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}
          then obtain b c where b:V=IntervalX(X,r,b,c) b∈Xc∈X by auto
          from b V(1) have m:⟨min,c⟩∈r⟨b,min⟩∈rmin≠b min≠c unfolding
IntervalX_def Interval_def by auto
          {
            assume A:⟨c,v⟩∈r
            from m obtain z where z:⟨z,c⟩∈r ⟨min,z⟩∈rz∈X-{c,min} us-
ing assms(4) unfolding IsDense_def
              using b(3) V(1,2) ‹U⊆X› by blast
            from z(2) have ⟨b,z⟩∈r using m(2) assms(1) unfolding IsLinOrder_def
trans_def
              by fast
            with z(1) have z∈IntervalX(X,r,b,c)∨z=b using z(3) unfold-
ing IntervalX_def
              Interval_def by auto
            then have z∈IntervalX(X,r,b,c) using m(2) z(2,3) using assms(1)
unfolding IsLinOrder_def
              antisym_def by auto
            with b(1) V(2) have z∈U by auto moreover
            from A z(1) have ⟨z,v⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast
            moreover have z≠v using A z(1,3) assms(1) unfolding IsLinOrder_def
antisym_def by auto
            ultimately have z∈U∩LeftRayX(X,r,v) unfolding LeftRayX_def
using z(3) by auto
            then have min∈r{z} using Order_ZF_4_L4(1)[OF _ Hmin] assms(1)
```

```
unfolding Supremum_def IsLinOrder_def
              by auto
            then have ⟨z,min⟩∈r by auto
            with z(2,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
          }
          then have vc:⟨v,c⟩∈rv≠c using assms(1) unfolding IsLinOrder_def
IsTotal_def using ‹v∈X-U›
            b(3) by auto
          {
            assume min=v
            with V(2,1) ‹v∈X-U› have False by auto
          }
          then have min≠v by auto
          with a1 obtain z where z:⟨min,z⟩∈r⟨z,v⟩∈rz∈X-{min,v} using
assms(4) unfolding IsDense_def
            using V(1,2) ‹U⊆X›‹v∈X-U› by blast
          from z(2) vc(1) have zc:⟨z,c⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def
            by fast moreover
          from m(2) z(1) have ⟨b,z⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def
            by fast ultimately
          have z∈Interval(r,b,c) using Order_ZF_2_L1B by auto moreover
          {
            assume z=c
            then have False using z(2) vc using assms(1) unfolding IsLinOrder_def
antisym_def
              by fast
          }
          then have z≠c by auto moreover
          {
            assume z=b
            then have z=min using m(2) z(1) using assms(1) unfolding IsLinOrder_def
              antisym_def by auto
            with z(3) have False by auto
          }
          then have z≠b by auto moreover
          have z∈X using z(3) by auto ultimately
          have z∈IntervalX(X,r,b,c) unfolding IntervalX_def by auto
          then have z∈V using b(1) by auto
          then have z∈U using V(2) by auto moreover
          from z(2,3) have z∈LeftRayX(X,r,v) unfolding LeftRayX_def by
auto ultimately
          have z∈U∩LeftRayX(X,r,v) by auto
          then have min∈r{z} using Order_ZF_4_L4(1)[OF _ Hmin] assms(1)
unfolding Supremum_def IsLinOrder_def
            by auto
          then have ⟨z,min⟩∈r by auto
```

```
          with z(1,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
        }
        ultimately have False using V(3) by auto
      }
      then have ass:min∈X-U using a1 assms(3) by auto
      then obtain V where V:min∈VV⊆X-U
        V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X} using point_open_base_neigh
        [OF Ordtopology_is_a_topology(2)[OF assms(1)] ‹X-U∈(OrdTopology
X r)› ass] by blast
      {
        assume V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}
        then obtain b c where b:V=IntervalX(X,r,b,c)b∈Xc∈X by auto
        from b V(1) have m:⟨min,c⟩∈r⟨b,min⟩∈rmin≠b min≠c unfolding IntervalX_def
Interval_def by auto
        {
          fix x assume A:x∈U∩LeftRayX(X,r,v)
          then have ⟨x,v⟩∈rx∈U unfolding LeftRayX_def by auto
          then have x∉V using V(2) by auto
          then have x∉Interval(r, b, c) ∩ X∨x=b∨x=c using b(1) unfold-
ing IntervalX_def by auto
          then have (⟨b,x⟩∉r∨⟨x,c⟩∉r)∨x=b∨x=cx∈X using Order_ZF_2_L1B
‹x∈U›‹U⊆X› by auto
          then have (⟨x,b⟩∈r∨⟨c,x⟩∈r)∨x=b∨x=c using assms(1) unfold-
ing IsLinOrder_def IsTotal_def
            using b(2,3) by auto
          then have (⟨x,b⟩∈r∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder_def
using total_is_refl
            unfolding refl_def using b(2,3) by auto moreover
          from A have ⟨x,min⟩∈r using Order_ZF_4_L4(1)[OF _ Hmin] assms(1)
unfolding Supremum_def IsLinOrder_def
            by auto
          ultimately have (⟨x,b⟩∈r∨⟨c,min⟩∈r) using assms(1) unfolding
IsLinOrder_def trans_def
            by fast
          with m(1) have (⟨x,b⟩∈r∨c=min) using assms(1) unfolding IsLinOrder_def
antisym_def by auto
          with m(4) have ⟨x,b⟩∈r by auto
        }
        then have ⟨min,b⟩∈r using Order_ZF_5_L3[OF _ nE Hmin] assms(1)
unfolding IsLinOrder_def by auto
        with m(2,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
      }
      moreover
      {
        assume V∈{RightRayX(X,r,b). b∈X}
        then obtain b where b:V=RightRayX(X,r,b) b∈X by auto
```

```
        from b V(1) have m:⟨b,min⟩∈rmin≠b unfolding RightRayX_def by
auto
        {
          fix x assume A:x∈U∩LeftRayX(X,r,v)
          then have ⟨x,v⟩∈rx∈U unfolding LeftRayX_def by auto
          then have x∉V using V(2) by auto
          then have x∉RightRayX(X,r, b) using b(1) by auto
          then have (⟨b,x⟩∉r∨x=b)x∈X unfolding RightRayX_def using ‹x∈U›‹U⊆X›
by auto
          then have ⟨x,b⟩∈r using assms(1) unfolding IsLinOrder_def us-
ing total_is_refl unfolding
            refl_def unfolding IsTotal_def using b(2) by auto
        }
        then have ⟨min,b⟩∈r using Order_ZF_5_L3[OF _ nE Hmin] assms(1)
unfolding IsLinOrder_def by auto
        with m(2,1) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
      } moreover
      {
         assume V∈{LeftRayX(X,r,b). b∈X}
        then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
        from b V(1) have m:⟨min,b⟩∈rmin≠b unfolding LeftRayX_def by auto
        {
          fix x assume A:x∈U∩LeftRayX(X,r,v)
          then have ⟨x,v⟩∈rx∈U unfolding LeftRayX_def by auto
          then have x∉V using V(2) by auto
          then have x∉LeftRayX(X,r, b) using b(1) by auto
          then have (⟨x,b⟩∉r∨x=b)x∈X unfolding LeftRayX_def using ‹x∈U›‹U⊆X›
by auto
          then have ⟨b,x⟩∈r using assms(1) unfolding IsLinOrder_def us-
ing total_is_refl unfolding
            refl_def unfolding IsTotal_def using b(2) by auto
          with m(1) have ⟨min,x⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast
          moreover
          from bound A have ∃g. ∀y∈U∩LeftRayX(X,r,v). ⟨y,g⟩∈r using
nE
            unfolding IsBoundedAbove_def by auto
          then obtain g where g:∀y∈U∩LeftRayX(X,r,v). ⟨y,g⟩∈r by auto
          with nE obtain t where t∈U∩LeftRayX(X,r,v) by auto
          with g have ⟨t,g⟩∈r by auto
          with assms(3) have g∈X by auto
          with g have boundX:∃g∈X. ∀y∈U∩LeftRayX(X,r,v). ⟨y,g⟩∈r by
auto
          have ⟨x,min⟩∈r using Order_ZF_5_L7(2)[OF assms(3) _ assms(5)
_  boundX]
            assms(1) ‹U⊆X› A unfolding LeftRayX_def IsLinOrder_def by
auto
          ultimately have x=min using assms(1) unfolding IsLinOrder_def
```

```
antisym_def by auto
        }
        then have U∩LeftRayX(X,r,v)⊆{min} by auto moreover
        {
          assume min∈U∩LeftRayX(X,r,v)
          then have min∈U by auto
          then have False using V(1,2) by auto
        }
        ultimately have False using nE by auto
      }
      moreover note V(3)
      ultimately have False by auto
    }
    with assms(1) have ⟨v,u⟩∈r unfolding IsLinOrder_def IsTotal_def us-
ing ‹u∈U›‹U⊆X›
      ‹v∈X-U› by auto
    have RightRayX(X,r,v)∈(OrdTopology X r) using base_sets_open[OF Ordtopology_is_a_topolo
assms(1)]]
      ‹v∈X-U› by auto
    then have U∩RightRayX(X,r,v)∈(OrdTopology X r) using U(3) using Ordtopology_is_a_topol
      [OF assms(1)] unfolding IsATopology_def by auto
    {
      fix b assume b∈(U)∩RightRayX(X,r,v)
      then have ⟨v,b⟩∈r unfolding RightRayX_def by auto
    }
    then have bound:IsBoundedBelow(U∩RightRayX(X,r,v),r) unfolding IsBoundedBelow_def
by auto
    with ‹⟨v,u⟩∈r›‹u∈U›‹U⊆X›‹v∈X-U› have nE:U∩RightRayX(X,r,v)≠0 un-
folding RightRayX_def by auto
    have Hmax:HasAmaximum(r,⋂c∈U∩RightRayX(X,r,v). r-{c}) using complete_order_bounded_bel
assms(5) bound nE assms(3)].
    let max=Infimum(r,U∩RightRayX(X,r,v))
    {
      fix c assume c∈U∩RightRayX(X,r,v)
      then have ⟨v,c⟩∈r unfolding RightRayX_def by auto
    }
    then have a1:⟨v,max⟩∈r using Order_ZF_5_L4[OF _ nE Hmax] assms(1)
unfolding IsLinOrder_def
      by auto
    {
      assume ass:max∈U
      then obtain V where V:max∈VV⊆U
        V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X} using point_open_base_neigh
        [OF Ordtopology_is_a_topology(2)[OF assms(1)] ‹U∈(OrdTopology
X r)› ass] by blast
      {
        assume V∈{RightRayX(X,r,b). b∈X}
        then obtain b where b:b∈X V=RightRayX(X,r,b) by auto
```

```
        from V(1) b(2) have a2:⟨b,max⟩∈rmax≠b unfolding RightRayX_def
by auto
        {
          assume ⟨b,v⟩∈r
          then have b=v∨v∈RightRayX(X,r,b) unfolding RightRayX_def us-
ing ‹v∈X-U› by auto
          then have b=v using b(2) V(2) ‹v∈X-U› by auto
        }
        then have bv:⟨v,b⟩∈r using assms(1) unfolding IsLinOrder_def IsTotal_def
using b(1)
          ‹v∈X-U› by auto
        from a2 assms(4) obtain z where z:⟨b,z⟩∈r⟨z,max⟩∈rz∈X-{b,max}
unfolding IsDense_def
          using b(1) V(1,2) ‹U⊆X› by blast
        then have rayb:z∈RightRayX(X,r,b) unfolding RightRayX_def by
auto
        from z(1) bv have ⟨v,z⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast moreover
        {
          assume z=v
          with bv have ⟨z,b⟩∈r by auto
          with z(1) have b=z using assms(1) unfolding IsLinOrder_def antisym_def
by auto
          then have False using z(3) by auto
        }
        ultimately have z∈RightRayX(X,r,v) unfolding RightRayX_def us-
ing z(3) by auto
        with rayb have z∈U∩RightRayX(X,r,v) using V(2) b(2) by auto
        then have max∈r-{z} using Order_ZF_4_L3(1)[OF _ Hmax] assms(1)
unfolding Infimum_def IsLinOrder_def
          by auto
        then have ⟨max,z⟩∈r by auto
        with z(2,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
      }
      moreover
      {
        assume V∈{LeftRayX(X,r,b). b∈X}
        then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
        note a1 moreover
        from V(1) b(1) have a2:⟨max,b⟩∈rmax≠b unfolding LeftRayX_def
by auto
        ultimately have ⟨v,b⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by blast moreover
        {
          assume b=v
          with a1 a2(1) have b=max using assms(1) unfolding IsLinOrder_def
antisym_def by auto
          with a2(2) have False by auto
```

```
        }
        ultimately have False using V(2) b(1) unfolding LeftRayX_def us-
ing ‹v∈X-U› by auto
      }
      moreover
      {
        assume V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}
        then obtain b c where b:V=IntervalX(X,r,b,c) b∈Xc∈X by auto
        from b V(1) have m:⟨max,c⟩∈r⟨b,max⟩∈rmax≠b max≠c unfolding IntervalX_def
Interval_def by auto
        {
          assume A:⟨v,b⟩∈r
          from m obtain z where z:⟨z,max⟩∈r ⟨b,z⟩∈rz∈X-{b,max} using
assms(4) unfolding IsDense_def
            using b(2) V(1,2) ‹U⊆X› by blast
          from z(1) have ⟨z,c⟩∈r using m(1) assms(1) unfolding IsLinOrder_def
trans_def
            by fast
          with z(2) have z∈IntervalX(X,r,b,c)∨z=c using z(3) unfold-
ing IntervalX_def
            Interval_def by auto
          then have z∈IntervalX(X,r,b,c) using m(1) z(1,3) using assms(1)
unfolding IsLinOrder_def
            antisym_def by auto
          with b(1) V(2) have z∈U by auto moreover
          from A z(2) have ⟨v,z⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def by fast
          moreover have z≠v using A z(2,3) assms(1) unfolding IsLinOrder_def
antisym_def by auto
          ultimately have z∈U∩RightRayX(X,r,v) unfolding RightRayX_def
using z(3) by auto
          then have max∈r-{z} using Order_ZF_4_L3(1)[OF _ Hmax] assms(1)
unfolding Infimum_def IsLinOrder_def
            by auto
          then have ⟨max,z⟩∈r by auto
          with z(1,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
        }
        then have vc:⟨b,v⟩∈rv≠b using assms(1) unfolding IsLinOrder_def
IsTotal_def using ‹v∈X-U›
          b(2) by auto
        {
          assume max=v
          with V(2,1) ‹v∈X-U› have False by auto
        }
        then have v≠max by auto moreover
        note a1 moreover
        have max∈X using V(1,2) ‹U⊆X› by auto
        moreover have v∈X using ‹v∈X-U› by auto
```

```
        ultimately obtain z where z:⟨v,z⟩∈r⟨z,max⟩∈rz∈X-{v,max} using
assms(4) unfolding IsDense_def
          by auto
        from z(1) vc(1) have zc:⟨b,z⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def
          by fast moreover
        from m(1) z(2) have ⟨z,c⟩∈r using assms(1) unfolding IsLinOrder_def
trans_def
          by fast ultimately
        have z∈Interval(r,b,c) using Order_ZF_2_L1B by auto moreover
        {
          assume z=b
          then have False using z(1) vc using assms(1) unfolding IsLinOrder_def
antisym_def
            by fast
        }
        then have z≠b by auto moreover
        {
          assume z=c
          then have z=max using m(1) z(2) using assms(1) unfolding IsLinOrder_def
            antisym_def by auto
          with z(3) have False by auto
        }
        then have z≠c by auto moreover
        have z∈X using z(3) by auto ultimately
        have z∈IntervalX(X,r,b,c) unfolding IntervalX_def by auto
        then have z∈V using b(1) by auto
        then have z∈U using V(2) by auto moreover
        from z(1,3) have z∈RightRayX(X,r,v) unfolding RightRayX_def by
auto ultimately
        have z∈U∩RightRayX(X,r,v) by auto
        then have max∈r-{z} using Order_ZF_4_L3(1)[OF _ Hmax] assms(1)
unfolding Infimum_def IsLinOrder_def
          by auto
        then have ⟨max,z⟩∈r by auto
        with z(2,3) have False using assms(1) unfolding IsLinOrder_def
antisym_def by auto
      }
      ultimately have False using V(3) by auto
    }
    then have ass:max∈X-U using a1 assms(3) by auto
    then obtain V where V:max∈VV⊆X-U
      V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}∪{LeftRayX(X,r,b). b∈X}∪{RightRayX(X,r,b).
b∈X} using point_open_base_neigh
      [OF Ordtopology_is_a_topology(2)[OF assms(1)] ‹X-U∈(OrdTopology
X r)› ass] by blast
    {
      assume V∈{IntervalX(X,r,b,c). ⟨b,c⟩∈X×X}
      then obtain b c where b:V=IntervalX(X,r,b,c)b∈Xc∈X by auto
```

```
      from b V(1) have m:⟨max,c⟩∈r⟨b,max⟩∈rmax≠b max≠c unfolding IntervalX_def
Interval_def by auto
      {
        fix x assume A:x∈U∩RightRayX(X,r,v)
        then have ⟨v,x⟩∈rx∈U unfolding RightRayX_def by auto
        then have x∉V using V(2) by auto
        then have x∉Interval(r, b, c) ∩ X∨x=b∨x=c using b(1) unfold-
ing IntervalX_def by auto
        then have (⟨b,x⟩∉r∨⟨x,c⟩∉r)∨x=b∨x=cx∈X using Order_ZF_2_L1B ‹x∈U›‹U⊆X›
by auto
        then have (⟨x,b⟩∈r∨⟨c,x⟩∈r)∨x=b∨x=c using assms(1) unfolding
IsLinOrder_def IsTotal_def
          using b(2,3) by auto
        then have (⟨x,b⟩∈r∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder_def
using total_is_refl
          unfolding refl_def using b(2,3) by auto moreover
        from A have ⟨max,x⟩∈r using Order_ZF_4_L3(1)[OF _ Hmax] assms(1)
unfolding Infimum_def IsLinOrder_def
          by auto
        ultimately have (⟨max,b⟩∈r∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder_def
trans_def
          by fast
        with m(2) have (max=b∨⟨c,x⟩∈r) using assms(1) unfolding IsLinOrder_def
antisym_def by auto
        with m(3) have ⟨c,x⟩∈r by auto
      }
      then have ⟨c,max⟩∈r using Order_ZF_5_L4[OF _ nE Hmax] assms(1) un-
folding IsLinOrder_def by auto
      with m(1,4) have False using assms(1) unfolding IsLinOrder_def antisym_def
by auto
    }
    moreover
    {
      assume V∈{RightRayX(X,r,b). b∈X}
      then obtain b where b:V=RightRayX(X,r,b) b∈X by auto
      from b V(1) have m:⟨b,max⟩∈rmax≠b unfolding RightRayX_def by auto
      {
        fix x assume A:x∈U∩RightRayX(X,r,v)
        then have ⟨v,x⟩∈rx∈U unfolding RightRayX_def by auto
        then have x∉V using V(2) by auto
        then have x∉RightRayX(X,r, b) using b(1) by auto
        then have (⟨b,x⟩∉r∨x=b)x∈X unfolding RightRayX_def using ‹x∈U›‹U⊆X›
by auto
        then have ⟨x,b⟩∈r using assms(1) unfolding IsLinOrder_def us-
ing total_is_refl unfolding
          refl_def unfolding IsTotal_def using b(2) by auto moreover
        from A have ⟨max,x⟩∈r using Order_ZF_4_L3(1)[OF _ Hmax] assms(1)
unfolding Infimum_def IsLinOrder_def
          by auto ultimately
```

```
        have ⟨max,b⟩∈r using assms(1) unfolding IsLinOrder_def trans_def
by fast
        with m have False using assms(1) unfolding IsLinOrder_def antisym_def
by auto
      }
      then have False using nE by auto
    } moreover
    {
       assume V∈{LeftRayX(X,r,b). b∈X}
      then obtain b where b:V=LeftRayX(X,r,b) b∈X by auto
      from b V(1) have m:⟨max,b⟩∈rmax≠b unfolding LeftRayX_def by auto
      {
        fix x assume A:x∈U∩RightRayX(X,r,v)
        then have ⟨v,x⟩∈rx∈U unfolding RightRayX_def by auto
        then have x∉V using V(2) by auto
        then have x∉LeftRayX(X,r, b) using b(1) by auto
        then have (⟨x,b⟩∉r∨x=b)x∈X unfolding LeftRayX_def using ‹x∈U›‹U⊆X›
by auto
        then have ⟨b,x⟩∈r using assms(1) unfolding IsLinOrder_def us-
ing total_is_refl unfolding
          refl_def unfolding IsTotal_def using b(2) by auto
        then have b∈r-{x} by auto
      }
      with nE have b∈(⋂c∈U∩RightRayX(X,r,v). r-{c}) by auto
      then have ⟨b,max⟩∈r unfolding Infimum_def using Order_ZF_4_L3(2)[OF
_ Hmax] assms(1)
        unfolding IsLinOrder_def by auto
      with m have False using assms(1) unfolding IsLinOrder_def antisym_def
by auto
    }
    moreover note V(3)
    ultimately have False by auto
  }
  then show thesis by auto
qed
```

87.4 Numerability axioms

A κ-separable order topology is in relation with order density.

If an order topology has a subset A which is topologically dense, then that subset is weakly order-dense in X.

```
lemma dense_top_imp_Wdense_ord:
  assumes IsLinOrder(X,r) Closure(A,OrdTopology X r)=X A⊆X ∃x y. x ≠
y ∧ x ∈ X ∧ y ∈ X
  shows A{is weakly dense in}X{with respect to}r
proof-
  {
    fix r1 r2 assume r1∈Xr2∈Xr1≠r2 ⟨r1,r2⟩∈r
```

```
    then have IntervalX(X,r,r1,r2)∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X
× X} ∪ {LeftRayX(X, r, b) . b ∈ X} ∪
      {RightRayX(X, r, b) . b ∈ X} by auto
    then have P:IntervalX(X,r,r1,r2)∈(OrdTopology X r) using base_sets_open[OF
Ordtopology_is_a_topology(2)[OF assms(1)]]
      by auto
    have IntervalX(X,r,r1,r2)⊆X unfolding IntervalX_def by auto
    then have int:Closure(A,OrdTopology X r)∩IntervalX(X,r,r1,r2)=IntervalX(X,r,r1,r2)
using assms(2) by auto
    {
      assume IntervalX(X,r,r1,r2)≠0
      then have A∩(IntervalX(X,r,r1,r2))≠0 using topology0.cl_inter_neigh[OF
topology0_ordtopology[OF assms(1)] _ P , of A]
        using assms(3) union_ordtopology[OF assms(1,4)] int by auto
    }
    then have (∃z∈A-{r1,r2}. ⟨r1,z⟩∈r∧⟨z,r2⟩∈r)∨IntervalX(X,r,r1,r2)=0
unfolding IntervalX_def
      Interval_def by auto
  }
  then show thesis unfolding IsWeaklyDenseSub_def by auto
qed
```

Conversely, a weakly order-dense set is topologically dense if it is also considered that: if there is a maximum or a minimum elements whose singletons are open, this points have to be in A. In conclusion, weakly order-density is a property closed to topological density.

Another way to see this: Consider a weakly order-dense set A:

- If X has a maximum and a minimum and $\{min, max\}$ is open: A is topologically dense in $X \setminus \{min, max\}$, where min is the minimum in X and max is the maximum in X.
- If X has a maximum, $\{max\}$ is open and X has no minimum or $\{min\}$ isn't open: A is topologically dense in $X \setminus \{max\}$, where max is the maximum in X.
- If X has a minimum, $\{min\}$ is open and X has no maximum or $\{max\}$ isn't open A is topologically dense in $X \setminus \{min\}$, where min is the minimum in X.
- If X has no minimum or maximum, or $\{min, max\}$ has no proper open sets: A is topologically dense in X.

```
lemma Wdense_ord_imp_dense_top:
  assumes IsLinOrder(X,r) A{is weakly dense in}X{with respect to}r A⊆X
∃x y. x ≠ y ∧ x ∈ X ∧ y ∈ X
    HasAminimum(r,X)⟶{Minimum(r,X)}∈(OrdTopology X r)⟶Minimum(r,X)∈A
    HasAmaximum(r,X)⟶{Maximum(r,X)}∈(OrdTopology X r)⟶Maximum(r,X)∈A
```

```
  shows Closure(A,OrdTopology X r)=X
proof-
  {
    fix x assume x∈X
  {
    fix U assume ass:x∈UU∈(OrdTopology X r)
    then have ∃V∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X} . V⊆U∧x∈V
      using point_open_base_neigh[OF Ordtopology_is_a_topology(2)[OF assms(1)]]
by auto
    then obtain V where V:V∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪
{LeftRayX(X, r, b) . b ∈ X} ∪ {RightRayX(X, r, b) . b ∈ X} V⊆U x∈V
      by blast
    note V(1) moreover
    {
      assume V∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X}
      then obtain b c where b:b∈Xc∈XV=IntervalX(X, r, b, c) by auto
      with V(3) have x:⟨b,x⟩∈r ⟨x,c⟩∈r x≠b x≠c unfolding IntervalX_def
Interval_def by auto
      then have ⟨b,c⟩∈r using assms(1) unfolding IsLinOrder_def trans_def
by fast
      moreover from x(1-3) have b≠c using assms(1) unfolding IsLinOrder_def
antisym_def by fast
      moreover note assms(2) b V(3)
      ultimately have ∃z∈A-{b,c}. ⟨b,z⟩∈r∧⟨z,c⟩∈r unfolding IsWeaklyDenseSub_def
by auto
      then obtain z where z∈Az≠bz≠c⟨b,z⟩∈r⟨z,c⟩∈r by auto
      with assms(3) have z∈Az∈IntervalX(X, r, b, c) unfolding IntervalX_def
Interval_def by auto
      then have A∩U≠0 using V(2) b(3) by auto
    }
    moreover
    {
      assume V∈{RightRayX(X, r, b) . b ∈ X}
      then obtain b where b:b∈XV=RightRayX(X, r, b) by auto
      with V(3) have x:⟨b,x⟩∈r b≠x unfolding RightRayX_def by auto more-
over
      note b(1) moreover
      have U⊆⋃(OrdTopology X r) using ass(2) by auto
      then have U⊆X using union_ordtopology[OF assms(1,4)] by auto
      then have x∈X using ass(1) by auto moreover
      note assms(2) ultimately
      have disj:(∃z∈A-{b,x}. ⟨b,z⟩∈r∧⟨z,x⟩∈r)∨ IntervalX(X, r, b, x)
= 0 unfolding IsWeaklyDenseSub_def by auto
      {
        assume B:IntervalX(X, r, b, x) = 0
        {
          assume ∃y∈X. ⟨x,y⟩∈r ∧ x≠y
          then obtain y where y:y∈X⟨x,y⟩∈r x≠y by auto
```

```
          with x have x∈IntervalX(X,r,b,y) unfolding IntervalX_def Interval_def
            using ‹x∈X› by auto moreover
          have ⟨b,y⟩∈r using y(2) x(1) assms(1) unfolding IsLinOrder_def
trans_def by fast
          moreover have b≠y using y(2,3) x(1) assms(1) unfolding IsLinOrder_def
antisym_def by fast
          ultimately
          have (∃z∈A-{b,y}. ⟨b,z⟩∈r∧⟨z,y⟩∈r) using assms(2) unfolding
IsWeaklyDenseSub_def
            using y(1) b(1) by auto
          then obtain z where z∈A⟨b,z⟩∈rb≠z by auto
          then have z∈A∩V using b(2) unfolding RightRayX_def using assms(3)
by auto
          then have z∈A∩U using V(2) by auto
          then have A∩U≠0 by auto
        }
        moreover
        {
          assume R:∀y∈X. ⟨x,y⟩∈r⟶x=y
          {
            fix y assume y∈RightRayX(X,r,b)
            then have y:⟨b,y⟩∈r y∈X-{b} unfolding RightRayX_def by auto
            {
              assume A:y≠x
              then have ⟨x,y⟩∉r using R y(2) by auto
              then have ⟨y,x⟩∈r using assms(1) unfolding IsLinOrder_def
IsTotal_def
                using ‹x∈X› y(2) by auto
              with A y have y∈IntervalX(X,r,b,x) unfolding IntervalX_def
Interval_def
                by auto
              then have False using B by auto
            }
            then have y=x by auto
          }
          then have RightRayX(X,r,b)={x} using V(3) b(2) by blast
          moreover
          {
            fix t assume T:t∈X
            {
              assume t=x
              then have ⟨t,x⟩∈r using assms(1) unfolding IsLinOrder_def
                using Order_ZF_1_L1 T by auto
            }
            moreover
            {
              assume t≠x
              then have ⟨x,t⟩∉r using R T by auto
              then have ⟨t,x⟩∈r using assms(1) unfolding IsLinOrder_def
```

```
IsTotal_def
                using T ‹x∈X› by auto
              }
              ultimately have ⟨t,x⟩∈r by auto
            }
            with ‹x∈X› have HM:HasAmaximum(r,X) unfolding HasAmaximum_def
by auto
            then have Maximum(r,X)∈X∀t∈X. ⟨t,Maximum(r,X)⟩∈r using Order_ZF_4_L3
assms(1) unfolding IsLinOrder_def
              by auto
            with R ‹x∈X› have xm:x=Maximum(r,X) by auto
            moreover note b(2)
            ultimately have V={Maximum(r,X)} by auto
            then have {Maximum(r,X)}∈(OrdTopology X r) using base_sets_open[OF
Ordtopology_is_a_topology(2)[OF assms(1)]]
              V(1) by auto
            with HM have Maximum(r,X)∈A using assms(6) by auto
            with xm have x∈A by auto
            with V(2,3) have A∩U≠0 by auto
          }
          ultimately have A∩U≠0 by auto
        }
        moreover
        {
          assume IntervalX(X, r, b, x) ≠ 0
          with disj have ∃z∈A-{b,x}. ⟨b,z⟩∈r∧⟨z,x⟩∈r by auto
          then obtain z where z∈Az≠b⟨b,z⟩∈r by auto
          then have z∈Az∈RightRayX(X,r,b) unfolding RightRayX_def using
assms(3) by auto
          then have z∈A∩U using V(2) b(2) by auto
          then have A∩U≠0 by auto
        }
        ultimately have A∩U≠0 by auto
      }
      moreover
      {
        assume V∈{LeftRayX(X, r, b) . b ∈ X}
        then obtain b where b:b∈XV=LeftRayX(X, r, b) by auto
        with V(3) have x:⟨x,b⟩∈r b≠x unfolding LeftRayX_def by auto more-
over
        note b(1) moreover
        have U⊆⋃(OrdTopology X r) using ass(2) by auto
        then have U⊆X using union_ordtopology[OF assms(1,4)] by auto
        then have x∈X using ass(1) by auto moreover
        note assms(2) ultimately
        have disj:(∃z∈A-{b,x}. ⟨x,z⟩∈r∧⟨z,b⟩∈r)∨ IntervalX(X, r, x, b)
= 0 unfolding IsWeaklyDenseSub_def by auto
        {
          assume B:IntervalX(X, r, x, b) = 0
```

```
        {
          assume ∃y∈X. ⟨y,x⟩∈r ∧ x≠y
          then obtain y where y:y∈X⟨y,x⟩∈r x≠y by auto
          with x have x∈IntervalX(X,r,y,b) unfolding IntervalX_def Interval_def
            using ‹x∈X› by auto moreover
          have ⟨y,b⟩∈r using y(2) x(1) assms(1) unfolding IsLinOrder_def
trans_def by fast
          moreover have b≠y using y(2,3) x(1) assms(1) unfolding IsLinOrder_def
antisym_def by fast
          ultimately
          have (∃z∈A-{b,y}. ⟨y,z⟩∈r∧⟨z,b⟩∈r) using assms(2) unfolding
IsWeaklyDenseSub_def
            using y(1) b(1) by auto
          then obtain z where z∈A⟨z,b⟩∈rb≠z by auto
          then have z∈A∩V using b(2) unfolding LeftRayX_def using assms(3)
by auto
          then have z∈A∩U using V(2) by auto
          then have A∩U≠0 by auto
        }
        moreover
        {
          assume R:∀y∈X. ⟨y,x⟩∈r⟶x=y
          {
            fix y assume y∈LeftRayX(X,r,b)
            then have y:⟨y,b⟩∈r y∈X-{b} unfolding LeftRayX_def by auto
            {
              assume A:y≠x
              then have ⟨y,x⟩∉r using R y(2) by auto
              then have ⟨x,y⟩∈r using assms(1) unfolding IsLinOrder_def
IsTotal_def
                using ‹x∈X› y(2) by auto
              with A y have y∈IntervalX(X,r,x,b) unfolding IntervalX_def
Interval_def
                by auto
              then have False using B by auto
            }
            then have y=x by auto
          }
          then have LeftRayX(X,r,b)={x} using V(3) b(2) by blast
          moreover
          {
            fix t assume T:t∈X
            {
              assume t=x
              then have ⟨x,t⟩∈r using assms(1) unfolding IsLinOrder_def
                using Order_ZF_1_L1 T by auto
            }
            moreover
            {
```

```
              assume t≠x
              then have ⟨t,x⟩∉r using R T by auto
              then have ⟨x,t⟩∈r using assms(1) unfolding IsLinOrder_def
IsTotal_def
                using T ‹x∈X› by auto
            }
            ultimately have ⟨x,t⟩∈r by auto
          }
          with ‹x∈X› have HM:HasAminimum(r,X) unfolding HasAminimum_def
by auto
          then have Minimum(r,X)∈X∀t∈X. ⟨Minimum(r,X),t⟩∈r using Order_ZF_4_L4
assms(1) unfolding IsLinOrder_def
            by auto
          with R ‹x∈X› have xm:x=Minimum(r,X) by auto
          moreover note b(2)
          ultimately have V={Minimum(r,X)} by auto
          then have {Minimum(r,X)}∈(OrdTopology X r) using base_sets_open[OF
Ordtopology_is_a_topology(2)[OF assms(1)]]
            V(1) by auto
          with HM have Minimum(r,X)∈A using assms(5) by auto
          with xm have x∈A by auto
          with V(2,3) have A∩U≠0 by auto
        }
        ultimately have A∩U≠0 by auto
      }
      moreover
      {
        assume IntervalX(X, r, x, b) ≠ 0
        with disj have ∃z∈A-{b,x}. ⟨x,z⟩∈r∧⟨z,b⟩∈r by auto
        then obtain z where z∈Az≠b⟨z,b⟩∈r by auto
        then have z∈Az∈LeftRayX(X,r,b) unfolding LeftRayX_def using assms(3)
by auto
        then have z∈A∩U using V(2) b(2) by auto
        then have A∩U≠0 by auto
      }
      ultimately have A∩U≠0 by auto
    }
    ultimately have A∩U≠0 by auto
  }
  then have ∀U∈(OrdTopology X r). x∈U ⟶ U∩A≠0 by auto
  moreover note ‹x∈X› moreover
  note assms(3) topology0.inter_neigh_cl[OF topology0_ordtopology[OF assms(1)]]
  union_ordtopology[OF assms(1,4)] ultimately have x∈Closure(A,OrdTopology
X r)
    by auto
  }
  then have X⊆Closure(A,OrdTopology X r) by auto
  with topology0.Top_3_L11(1)[OF topology0_ordtopology[OF assms(1)]]
    assms(3) union_ordtopology[OF assms(1,4)] show thesis by auto
```

```
qed
```

The conclusion is that an order topology is κ-separable iff there is a set A with cardinality strictly less than κ which is weakly-dense in X.

```
theorem separable_imp_wdense:
  assumes (OrdTopology X r){is separable of cardinal}Q ∃x y. x ≠ y ∧
x ∈ X ∧ y ∈ X
    IsLinOrder(X,r)
  shows ∃A∈Pow(X). A≺Q ∧ (A{is weakly dense in}X{with respect to}r)
proof-
  from assms obtain U where U∈Pow(⋃(OrdTopology X r)) Closure(U,OrdTopology
X r)=⋃(OrdTopology X r) U≺Q
    unfolding IsSeparableOfCard_def by auto
  then have U∈Pow(X) Closure(U,OrdTopology X r)=X U≺Q using union_ordtopology[OF
assms(3,2)]
    by auto
  with dense_top_imp_Wdense_ord[OF assms(3) _ _ assms(2)] show thesis
by auto
qed

theorem wdense_imp_separable:
  assumes ∃x y. x ≠ y ∧ x ∈ X ∧ y ∈ X (A{is weakly dense in}X{with
respect to}r)
    IsLinOrder(X,r) A≺Q InfCard(Q) A⊆X
  shows (OrdTopology X r){is separable of cardinal}Q
proof-
  {
    assume Hmin:HasAmaximum(r,X)
    then have MaxX:Maximum(r,X)∈X using Order_ZF_4_L3(1) assms(3) un-
folding IsLinOrder_def
      by auto
    {
      assume HMax:HasAminimum(r,X)
      then have MinX:Minimum(r,X)∈X using Order_ZF_4_L4(1) assms(3) un-
folding IsLinOrder_def
        by auto
      let A=A ∪{Maximum(r,X),Minimum(r,X)}
      have Finite({Maximum(r,X),Minimum(r,X)}) by auto
      then have {Maximum(r,X),Minimum(r,X)}≺nat using n_lesspoll_nat
        unfolding Finite_def using eq_lesspoll_trans by auto
      moreover
      from assms(5) have nat≺Q∨nat=Q unfolding InfCard_def
        using lt_Card_imp_lesspoll[of Qnat] unfolding lt_def succ_def
        using Card_is_Ord[of Q] by auto
      ultimately have {Maximum(r,X),Minimum(r,X)}≺Q using lesspoll_trans
by auto
      with assms(4,5) have C:A≺Q using less_less_imp_un_less
        by auto
      have WeakDense:A{is weakly dense in}X{with respect to}r using assms(2)
```

```
unfolding
        IsWeaklyDenseSub_def by auto
      from MaxX MinX assms(6) have S:A⊆X by auto
      then have Closure(A,OrdTopology X r)=X using Wdense_ord_imp_dense_top
        [OF assms(3) WeakDense _ assms(1)] by auto
      then have thesis unfolding IsSeparableOfCard_def using union_ordtopology[OF
assms(3,1)]
        S C by auto
    }
    moreover
    {
      assume nmin:¬HasAminimum(r,X)
       let A=A ∪{Maximum(r,X)}
      have Finite({Maximum(r,X)}) by auto
      then have {Maximum(r,X)}≺nat using n_lesspoll_nat
        unfolding Finite_def using eq_lesspoll_trans by auto
      moreover
      from assms(5) have nat≺Q∨nat=Q unfolding InfCard_def
        using lt_Card_imp_lesspoll[of Qnat] unfolding lt_def succ_def
        using Card_is_Ord[of Q] by auto
      ultimately have {Maximum(r,X)}≺Q using lesspoll_trans by auto
      with assms(4,5) have C:A≺Q using less_less_imp_un_less
        by auto
      have WeakDense:A{is weakly dense in}X{with respect to}r using assms(2)
unfolding
        IsWeaklyDenseSub_def by auto
      from MaxX assms(6) have S:A⊆X by auto
      then have Closure(A,OrdTopology X r)=X using Wdense_ord_imp_dense_top
        [OF assms(3) WeakDense _ assms(1)] nmin by auto
      then have thesis unfolding IsSeparableOfCard_def using union_ordtopology[OF
assms(3,1)]
        S C by auto
    }
    ultimately have thesis by auto
  }
  moreover
  {
    assume nmax:¬HasAmaximum(r,X)
    {
      assume HMin:HasAminimum(r,X)
      then have MinX:Minimum(r,X)∈X using Order_ZF_4_L4(1) assms(3) un-
folding IsLinOrder_def
        by auto
      let A=A ∪{Minimum(r,X)}
      have Finite({Minimum(r,X)}) by auto
      then have {Minimum(r,X)}≺nat using n_lesspoll_nat
        unfolding Finite_def using eq_lesspoll_trans by auto
      moreover
      from assms(5) have nat≺Q∨nat=Q unfolding InfCard_def
```

```
        using lt_Card_imp_lesspoll[of Qnat] unfolding lt_def succ_def
        using Card_is_Ord[of Q] by auto
      ultimately have {Minimum(r,X)}≺Q using lesspoll_trans by auto
      with assms(4,5) have C:A≺Q using less_less_imp_un_less
        by auto
      have WeakDense:A{is weakly dense in}X{with respect to}r using assms(2)
unfolding
        IsWeaklyDenseSub_def by auto
      from MinX assms(6) have S:A⊆X by auto
      then have Closure(A,OrdTopology X r)=X using Wdense_ord_imp_dense_top
        [OF assms(3) WeakDense _ assms(1)] nmax by auto
      then have thesis unfolding IsSeparableOfCard_def using union_ordtopology[OF
assms(3,1)]
        S C by auto
    }
    moreover
    {
      assume nmin:¬HasAminimum(r,X)
      let A=A
      from assms(4,5) have C:A≺Q by auto
      have WeakDense:A{is weakly dense in}X{with respect to}r using assms(2)
unfolding
        IsWeaklyDenseSub_def by auto
      from assms(6) have S:A⊆X by auto
      then have Closure(A,OrdTopology X r)=X using Wdense_ord_imp_dense_top
        [OF assms(3) WeakDense _ assms(1)] nmin nmax by auto
      then have thesis unfolding IsSeparableOfCard_def using union_ordtopology[OF
assms(3,1)]
        S C by auto
    }
    ultimately have thesis by auto
  }
  ultimately show thesis by auto
qed


end
```

88 Properties in topology 2

```
theory Topology_ZF_properties_2 imports Topology_ZF_7 Topology_ZF_1b
  Finite_ZF_1 Topology_ZF_11

begin
```

88.1 Local properties.

This theory file deals with local topological properties; and applies local compactness to the one point compactification.

We will say that a topological space is locally @term"P" iff every point has a neighbourhood basis of subsets that have the property @term"P" as subspaces.

```
definition
  IsLocally (_{is locally}_ 90)
  where T{is a topology} ⟹ T{is locally}P ≡ (∀x∈⋃T. ∀b∈T. x∈b ⟶
(∃c∈Pow(b). x∈Interior(c,T) ∧ P(c,T)))
```

88.2 First examples

Our first examples deal with the locally finite property. Finiteness is a property of sets, and hence it is preserved by homeomorphisms; which are in particular bijective.

The discrete topology is locally finite.

```
lemma discrete_locally_finite:
  shows Pow(A){is locally}(λA.(λB. Finite(A)))
proof-
  have ∀b∈Pow(A). ⋃(Pow(A){restricted to}b)=b unfolding RestrictedTo_def
by blast
  then have ∀b∈{{x}. x∈A}. Finite(b) by auto moreover
  have reg:∀S∈Pow(A). Interior(S,Pow(A))=S unfolding Interior_def by
auto
  {
    fix x b assume x∈⋃Pow(A) b∈Pow(A) x∈b
    then have {x}⊆b x∈Interior({x},Pow(A)) Finite({x}) using reg by
auto
    then have ∃c∈Pow(b). x∈Interior(c,Pow(A))∧Finite(c) by blast
  }
  then have ∀x∈⋃Pow(A). ∀b∈Pow(A). x∈b ⟶ (∃c∈Pow(b). x∈Interior(c,Pow(A))
∧ Finite(c)) by auto
  then show thesis using IsLocally_def[OF Pow_is_top] by auto
qed
```

The included set topology is locally finite when the set is finite.

```
lemma included_finite_locally_finite:
  assumes Finite(A) and A⊆X
  shows (IncludedSet(X,A)){is locally}(λA.(λB. Finite(A)))
proof-
  have ∀b∈Pow(X). b∩A⊆b by auto moreover
  note assms(1)
  ultimately have rr:∀b∈{A∪{x}. x∈X}. Finite(b) by force
  {
```

```
    fix x b assume x∈⋃(IncludedSet(X,A)) b∈(IncludedSet(X,A)) x∈b
    then have A∪{x}⊆b A∪{x}∈{A∪{x}. x∈X} and sub: b⊆X unfolding IncludedSet_def
by auto
    moreover have A ∪ {x} ⊆ X using assms(2) sub ‹x∈b› by auto
    then have x∈Interior(A∪{x},IncludedSet(X,A)) using interior_set_includedset[of
A∪{x}XA] by auto
    ultimately have ∃c∈Pow(b). x∈Interior(c,IncludedSet(X,A))∧ Finite(c)
using rr by blast
  }
  then have ∀x∈⋃(IncludedSet(X,A)). ∀b∈(IncludedSet(X,A)). x∈b ⟶
(∃c∈Pow(b). x∈Interior(c,IncludedSet(X,A))∧ Finite(c)) by auto
  then show thesis using IsLocally_def includedset_is_topology by auto
qed
```

88.3 Local compactness

```
definition
  IsLocallyComp (_{is locally-compact} 70)
  where T{is locally-compact}≡T{is locally}(λB. λT. B{is compact in}T)
```

We center ourselves in local compactness, because it is a very important tool in topological groups and compactifications.

If a subset is compact of some cardinal for a topological space, it is compact of the same cardinal in the subspace topology.

```
lemma compact_imp_compact_subspace:
  assumes A{is compact of cardinal}K{in}T A⊆B
  shows A{is compact of cardinal}K{in}(T{restricted to}B) unfolding IsCompactOfCard_def
proof
  from assms show C:Card(K) unfolding IsCompactOfCard_def by auto
  from assms have A⊆⋃T unfolding IsCompactOfCard_def by auto
  then have AA:A⊆⋃(T{restricted to}B) using assms(2) unfolding RestrictedTo_def
by auto moreover
  {
    fix M assume M∈Pow(T{restricted to}B) A⊆⋃M
    let M={S∈T. B∩S∈M}
    from ‹M∈Pow(T{restricted to}B)› have ⋃M⊆⋃M unfolding RestrictedTo_def
by auto
    with ‹A⊆⋃M› have A⊆⋃MM∈Pow(T) by auto
    with assms have ∃N∈Pow(M). A⊆⋃N∧N≺K unfolding IsCompactOfCard_def
by auto
    then obtain N where N∈Pow(M) A⊆⋃N N≺K by auto
    then have N{restricted to}B⊆M unfolding RestrictedTo_def FinPow_def
by auto
    moreover
    let f={⟨𝔅,B∩𝔅⟩. 𝔅∈N}
    have f:N→(N{restricted to}B) unfolding Pi_def function_def domain_def
RestrictedTo_def by auto
    then have f∈surj(N,N{restricted to}B) unfolding surj_def RestrictedTo_def
using apply_equality
```

```
      by auto
    from ‹N≺K› have N≲K unfolding lesspoll_def by auto
    with ‹f∈surj(N,N{restricted to}B)› have N{restricted to}B≲N using
surj_fun_inv_2 Card_is_Ord C by auto
    with ‹N≺K› have N{restricted to}B≺K using lesspoll_trans1 by auto
    moreover from ‹A⊆⋃N› have A⊆⋃(N{restricted to}B) using assms(2)
unfolding RestrictedTo_def by auto
    ultimately have ∃N∈Pow(M). A⊆⋃N ∧ N≺K by auto
  }
  with AA show A ⊆ ⋃(T {restricted to} B) ∧ (∀M∈Pow(T {restricted to}
B). A ⊆ ⋃M ⟶ (∃N∈Pow(M). A ⊆ ⋃N ∧ N≺K)) by auto
qed
```

The converse of the previous result is not always true. For compactness, it holds because the axiom of finite choice always holds.

```
lemma compact_subspace_imp_compact:
  assumes A{is compact in}(T{restricted to}B) A⊆B
  shows A{is compact in}T unfolding IsCompact_def
proof
  from assms show A⊆⋃T unfolding IsCompact_def RestrictedTo_def by
auto
next
  {
    fix M assume M∈Pow(T) A⊆⋃M
    let M=M{restricted to}B
    from ‹M∈Pow(T)› have M∈Pow(T{restricted to}B) unfolding RestrictedTo_def
by auto
    from ‹A⊆⋃M› have A⊆⋃M unfolding RestrictedTo_def using assms(2)
by auto
    with assms ‹M∈Pow(T{restricted to}B)› obtain N where N∈FinPow(M)
A⊆⋃N unfolding IsCompact_def by blast
    from ‹N∈FinPow(M)› have N≺nat unfolding FinPow_def Finite_def us-
ing n_lesspoll_nat eq_lesspoll_trans
      by auto
    then have Finite(N) using lesspoll_nat_is_Finite by auto
    then obtain n where n∈nat N≈n unfolding Finite_def by auto
    then have N≲n using eqpoll_imp_lepoll by auto
    moreover
    {
      fix BB assume BB∈N
      with ‹N∈FinPow(M)› have BB∈M unfolding FinPow_def by auto
      then obtain S where S∈M and BB=B∩S unfolding RestrictedTo_def
by auto
      then have S∈{S∈M. B∩S=BB} by auto
      then obtain {S∈M. B∩S=BB}≠0 by auto
    }
    then have ∀BB∈N. ((λW∈N. {S∈M. B∩S=W})BB)≠0 by auto moreover
    from ‹n∈nat› have  (N ≲ n ∧ (∀t∈N. (λW∈N. {S∈M. B∩S=W})  t ≠ 0)
⟶ (∃f. f ∈ Pi(N,λt. (λW∈N. {S∈M. B∩S=W})  t) ∧ (∀t∈N. f  t ∈ (λW∈N.
```

```
{S∈M. B∩S=W})  t))) using finite_choice unfolding AxiomCardinalChoiceGen_def
by blast
    ultimately
    obtain f where AA:f∈Pi(N,λt. (λW∈N. {S∈M. B∩S=W})  t) ∀t∈N. ft∈(λW∈N.
{S∈M. B∩S=W})  t by blast
    from AA(2) have ss:∀t∈N. ft∈{S∈M. B∩S=t} using beta_if by auto
    then have {ft. t∈N}⊆M by auto
    {
      fix t assume t∈N
      with ss have ft∈{S∈M. B∩S∈N} by auto
    }
    with AA(1) have FF:f:N→{S∈M. B∩S∈N} unfolding Pi_def Sigma_def us-
ing beta_if by auto moreover
    {
      fix aa bb assume AAA:aa∈N bb∈N faa=fbb
      from AAA(1) ss have B∩ (faa) =aa by auto
      with AAA(3) have B∩(fbb)=aa by auto
      with ss AAA(2) have aa=bb by auto
    }
    ultimately have f∈inj(N,{S∈M. B∩S∈N}) unfolding inj_def by auto
    then have f∈bij(N,range(f)) using inj_bij_range by auto
    then have f∈bij(N,fN) using range_image_domain FF by auto
    then have f∈bij(N,{ft. t∈N}) using func_imagedef FF by auto
    then have N≈{ft. t∈N} unfolding eqpoll_def by auto
    with ‹N≈n› have {ft. t∈N}≈n using eqpoll_sym eqpoll_trans by blast
    with ‹n∈nat› have Finite({ft. t∈N}) unfolding Finite_def by auto
    with ss have {ft. t∈N}∈FinPow(M) unfolding FinPow_def by auto more-
over
    {
      fix aa assume aa∈A
      with ‹A⊆⋃N› obtain b where b∈N and aa∈b by auto
      with ss have B∩(fb)=b by auto
      with ‹aa∈b› have aa∈B∩(fb) by auto
      then have aa∈ fb by auto
      with ‹b∈N› have aa∈⋃{ft. t∈N} by auto
    }
    then have A⊆⋃{ft. t∈N} by auto ultimately
    have ∃R∈FinPow(M). A⊆⋃R by auto
  }
  then show ∀M∈Pow(T). A ⊆ ⋃M ⟶ (∃N∈FinPow(M). A ⊆ ⋃N) by auto
qed
```

If the axiom of choice holds for some cardinal, then we can drop the compact sets of that cardial are compact of the same cardinal as subspaces of every superspace.

```
lemma Kcompact_subspace_imp_Kcompact:
  assumes A{is compact of cardinal}Q{in}(T{restricted to}B) A⊆B ({the
axiom of} Q {choice holds})
  shows A{is compact of cardinal}Q{in}T
```

```
proof -
  from assms(1) have a1:Card(Q) unfolding IsCompactOfCard_def RestrictedTo_def
by auto
  from assms(1) have a2:A⊆⋃T unfolding IsCompactOfCard_def RestrictedTo_def
by auto
  {
    fix M assume M∈Pow(T) A⊆⋃M
    let M=M{restricted to}B
    from ‹M∈Pow(T)› have M∈Pow(T{restricted to}B) unfolding RestrictedTo_def
by auto
    from ‹A⊆⋃M› have A⊆⋃M unfolding RestrictedTo_def using assms(2)
by auto
    with assms ‹M∈Pow(T{restricted to}B)› obtain N where N:N∈Pow(M)
A⊆⋃N N ≺ Q unfolding IsCompactOfCard_def by blast
    from N(3) have N≲Q using lesspoll_imp_lepoll by auto moreover
    {
      fix BB assume BB∈N
      with ‹N∈Pow(M)› have BB∈M unfolding FinPow_def by auto
      then obtain S where S∈M and BB=B∩S unfolding RestrictedTo_def
by auto
      then have S∈{S∈M. B∩S=BB} by auto
      then obtain {S∈M. B∩S=BB}≠0 by auto
    }
    then have ∀BB∈N. ((λW∈N. {S∈M. B∩S=W})BB)≠0 by auto moreover
    have  (N ≲ Q ∧ (∀t∈N. (λW∈N. {S∈M. B∩S=W})  t ≠ 0) ⟶ (∃f. f ∈
Pi(N,λt. (λW∈N. {S∈M. B∩S=W})  t) ∧ (∀t∈N. f  t ∈ (λW∈N. {S∈M. B∩S=W})
t)))
      using assms(3) unfolding AxiomCardinalChoiceGen_def by blast
    ultimately
    obtain f where AA:f∈Pi(N,λt. (λW∈N. {S∈M. B∩S=W})  t) ∀t∈N. ft∈(λW∈N.
{S∈M. B∩S=W})  t by blast
    from AA(2) have ss:∀t∈N. ft∈{S∈M. B∩S=t} using beta_if by auto
    then have {ft. t∈N}⊆M by auto
    {
      fix t assume t∈N
      with ss have ft∈{S∈M. B∩S∈N} by auto
    }
    with AA(1) have FF:f:N→{S∈M. B∩S∈N} unfolding Pi_def Sigma_def us-
ing beta_if by auto moreover
    {
      fix aa bb assume AAA:aa∈N bb∈N faa=fbb
      from AAA(1) ss have B∩ (faa) =aa by auto
      with AAA(3) have B∩(fbb)=aa by auto
      with ss AAA(2) have aa=bb by auto
    }
    ultimately have f∈inj(N,{S∈M. B∩S∈N}) unfolding inj_def by auto
    then have f∈bij(N,range(f)) using inj_bij_range by auto
    then have f∈bij(N,fN) using range_image_domain FF by auto
    then have f∈bij(N,{ft. t∈N}) using func_imagedef FF by auto
```

```
    then have N≈{ft. t∈N} unfolding eqpoll_def by auto
    with ‹N≺Q› have {ft. t∈N}≺Q using eqpoll_sym eq_lesspoll_trans by
blast moreover
    with ss have {ft. t∈N}∈Pow(M) unfolding FinPow_def by auto more-
over
    {
      fix aa assume aa∈A
      with ‹A⊆⋃N› obtain b where b∈N and aa∈b by auto
      with ss have B∩(fb)=b by auto
      with ‹aa∈b› have aa∈B∩(fb) by auto
      then have aa∈ fb by auto
      with ‹b∈N› have aa∈⋃{ft. t∈N} by auto
    }
    then have A⊆⋃{ft. t∈N} by auto ultimately
    have ∃R∈Pow(M). A⊆⋃R ∧ R≺Q by auto
  }
  then show thesis using a1 a2 unfolding IsCompactOfCard_def by auto
qed
```

Every set, with the cofinite topology is compact.

```
lemma cofinite_compact:
  shows X {is compact in}(CoFinite X) unfolding IsCompact_def
proof
  show X⊆⋃(CoFinite X) using union_cocardinal unfolding Cofinite_def
by auto
next
  {
    fix M assume M∈Pow(CoFinite X) X⊆⋃M
    {
      assume M=0∨M={0}
      then have M∈FinPow(M) unfolding FinPow_def by auto
      with ‹X⊆⋃M› have ∃N∈FinPow(M). X⊆⋃N by auto
    }
    moreover
    {
      assume M≠0M≠{0}
      then obtain U where U∈MU≠0 by auto
      with ‹M∈Pow(CoFinite X)› have U∈CoFinite X by auto
      with ‹U≠0› have U⊆X (X-U)≺nat unfolding Cofinite_def CoCardinal_def
by auto
      then have Finite(X-U) using lesspoll_nat_is_Finite by auto
      then have (X-U){is in the spectrum of}(λT. (⋃T){is compact in}T)
using compact_spectrum
        by auto
      then have ((⋃(CoFinite (X-U)))≈X-U) ⟶ ((⋃(CoFinite (X-U))){is
compact in}(CoFinite (X-U))) unfolding Spec_def
        using InfCard_nat CoCar_is_topology unfolding Cofinite_def by
auto
      then have com:(X-U){is compact in}(CoFinite (X-U)) using union_cocardinal
```

```
unfolding Cofinite_def by auto
      have (X-U)∩X=X-U by auto
      then have (CoFinite X){restricted to}(X-U)=(CoFinite (X-U)) us-
ing subspace_cocardinal unfolding Cofinite_def by auto
      with com have (X-U){is compact in}(CoFinite X) using compact_subspace_imp_compact[of
X-UCoFinite XX-U] by auto
      moreover have X-U⊆⋃M using ‹X⊆⋃M› by auto
      moreover note ‹M∈Pow(CoFinite X)›
      ultimately have ∃N∈FinPow(M). X-U⊆⋃N unfolding IsCompact_def by
auto
      then obtain N where N⊆M Finite(N) X-U⊆⋃N unfolding FinPow_def
by auto
      with ‹U∈M› have N ∪{U}⊆M Finite(N ∪{U}) X⊆⋃(N ∪{U}) by auto
      then have ∃N∈FinPow(M). X⊆⋃N unfolding FinPow_def by blast
    }
    ultimately
    have ∃N∈FinPow(M). X⊆⋃N by auto
  }
  then show ∀M∈Pow(CoFinite X). X ⊆ ⋃M ⟶ (∃N∈FinPow(M). X ⊆ ⋃N)
by auto
qed
```

A corollary is then that the cofinite topology is locally compact; since every subspace of a cofinite space is cofinite.

```
corollary cofinite_locally_compact:
  shows (CoFinite X){is locally-compact}
proof-
  have cof:topology0(CoFinite X) and cof1:(CoFinite X){is a topology}

    using CoCar_is_topology InfCard_nat Cofinite_def unfolding topology0_def
by auto
  {
    fix x B assume x∈⋃(CoFinite X) B∈(CoFinite X) x∈B
    then have x∈Interior(B,CoFinite X) using topology0.Top_2_L3[OF cof]
by auto moreover
    from ‹B∈(CoFinite X)› have B⊆X unfolding Cofinite_def CoCardinal_def
by auto
    then have B∩X=B by auto
    then have (CoFinite X){restricted to}B=CoFinite B using subspace_cocardinal
unfolding Cofinite_def by auto
    then have B{is compact in}((CoFinite X){restricted to}B) using cofinite_compact
      union_cocardinal unfolding Cofinite_def by auto
    then have B{is compact in}(CoFinite X) using compact_subspace_imp_compact
by auto
    ultimately have ∃c∈Pow(B). x∈Interior(c,CoFinite X)∧ c{is compact
in}(CoFinite X) by auto
  }
  then have (∀x∈⋃(CoFinite X). ∀b∈(CoFinite X). x∈b ⟶ (∃c∈Pow(b).
x∈Interior(c,CoFinite X) ∧ c{is compact in}(CoFinite X)))
```

```
    by auto
  then show thesis unfolding IsLocallyComp_def IsLocally_def[OF cof1]
by auto
qed
```

In every locally compact space, by definition, every point has a compact neighbourhood.

```
theorem (in topology0) locally_compact_exist_compact_neig:
  assumes T{is locally-compact}
  shows ∀x∈⋃T. ∃A∈Pow(⋃T). A{is compact in}T ∧ x∈int(A)
proof-
  {
    fix x assume x∈⋃T moreover
    then have ⋃T≠0 by auto
    have ⋃T∈T using union_open topSpaceAssum by auto
    ultimately have ∃c∈Pow(⋃T). x∈int(c)∧ c{is compact in}T using assms

      IsLocally_def topSpaceAssum unfolding IsLocallyComp_def by auto
    then have ∃c∈Pow(⋃T). c{is compact in}T ∧ x∈int(c) by auto
  }
  then show thesis by auto
qed
```

In Hausdorff spaces, the previous result is an equivalence.

```
theorem (in topology0) exist_compact_neig_T2_imp_locally_compact:
  assumes ∀x∈⋃T. ∃A∈Pow(⋃T). x∈int(A) ∧ A{is compact in}T T{is T2}
  shows T{is locally-compact}
proof-
  {
    fix x assume x∈⋃T
    with assms(1) obtain A where A∈Pow(⋃T) x∈int(A) and Acom:A{is compact
in}T by blast
    then have Acl:A{is closed in}T using in_t2_compact_is_cl assms(2)
by auto
    then have sub:A⊆⋃T unfolding IsClosed_def by auto
    {
      fix U assume U∈T x∈U
      let V=int(A∩U)
      from ‹x∈U› ‹x∈int(A)› have x∈U∩(int (A)) by auto
      moreover from ‹U∈T› have U∩(int(A))∈T using Top_2_L2 topSpaceAssum
unfolding IsATopology_def
        by auto moreover
      have U∩(int(A))⊆A∩U using Top_2_L1 by auto
      ultimately have x∈V using Top_2_L5 by blast
      have V⊆A using Top_2_L1 by auto
      then have cl(V)⊆A using Acl Top_3_L13 by auto
      then have A∩cl(V)=cl(V) by auto moreover
      have clcl:cl(V){is closed in}T using cl_is_closed(1) ‹V⊆A› ‹A⊆⋃T›
by auto
```

ultimately have comp:cl(V){is compact in}T **using** Acom compact_closed[of AnatTcl(V)] Compact_is_card_nat
by auto

{

then have cl(V){is compact in}(T{restricted to}cl(V)) **using** compact_imp_compact_sub
cl(V)natT] Compact_is_card_nat
by auto **moreover**

have ⋃(T{restricted to}cl(V))=cl(V) **unfolding** RestrictedTo_def **using** clcl **unfolding** IsClosed_def **by** auto **moreover**

ultimately have (⋃(T{restricted to}cl(V))){is compact in}(T{restricted to}cl(V)) **by** auto

}

then have (⋃(T{restricted to}cl(V))){is compact in}(T{restricted to}cl(V)) **by** auto **moreover**

have (T{restricted to}cl(V)){is T_2} **using** assms(2) T2_here clcl **unfolding** IsClosed_def **by** auto

ultimately have (T{restricted to}cl(V)){is T_4} **using** topology0.T2_compact_is_normal **unfolding** topology0_def
using Top_1_L4 **unfolding** isT4_def **using** T2_is_T1 **by** auto

then have clvreg:(T{restricted to}cl(V)){is regular} **using** topology0.T4_is_T3 **unfolding** topology0_def isT3_def **using** Top_1_L4
by auto

have V⊆cl(V) **using** cl_contains_set ‹V⊆A› ‹A⊆⋃T› **by** auto

then have V∈(T{restricted to}cl(V)) **unfolding** RestrictedTo_def **using** Top_2_L2 **by** auto

with ‹x∈V› **obtain** W **where** Wop:W∈(T{restricted to}cl(V)) **and** clcont:Closure(W,(T{res
to}cl(V)))⊆V **and** cinW:x∈W
using topology0.regular_imp_exist_clos_neig **unfolding** topology0_def **using** Top_1_L4 clvreg
by blast

from clcont Wop **have** W⊆V **using** topology0.cl_contains_set **unfolding** topology0_def **using** Top_1_L4 **by** auto

with Wop **have** W∈(T{restricted to}cl(V)){restricted to}V **unfolding** RestrictedTo_def **by** auto

moreover from ‹V⊆A› ‹A⊆⋃T› **have** V⊆⋃T **by** auto

then have V⊆cl(V)cl(V)⊆⋃T **using** ‹V⊆cl(V)› Top_3_L11(1) **by** auto

then have (T{restricted to}cl(V)){restricted to}V=(T{restricted to}V) **using** subspace_of_subspace **by** auto

ultimately have W∈(T{restricted to}V) **by** auto

then obtain UU **where** UU∈T W=UU∩V **unfolding** RestrictedTo_def **by** auto

then have W∈T **using** Top_2_L2 topSpaceAssum **unfolding** IsATopology_def **by** auto **moreover**

have W⊆Closure(W,(T{restricted to}cl(V))) **using** topology0.cl_contains_set **unfolding** topology0_def
using Top_1_L4 Wop **by** auto

ultimately have A1:x∈int(Closure(W,(T{restricted to}cl(V)))) **using** Top_2_L6 cinW **by** auto

from clcont **have** A2:Closure(W,(T{restricted to}cl(V)))⊆U **using**

```
Top_2_L1 by auto
      have clwcl:Closure(W,(T{restricted to}cl(V))) {is closed in}(T{restricted
to}cl(V))
        using topology0.cl_is_closed(1) Top_1_L4 Wop unfolding topology0_def
by auto
      from comp have cl(V){is compact in}(T{restricted to}cl(V)) us-
ing compact_imp_compact_subspace[of cl(V)natT] Compact_is_card_nat
          by auto
      with clwcl have ((cl(V)∩(Closure(W,(T{restricted to}cl(V)))))){is
compact in}(T{restricted to}cl(V))
        using compact_closed Compact_is_card_nat by auto moreover
      from clcont have cont:(Closure(W,(T{restricted to}cl(V))))⊆cl(V)
using cl_contains_set ‹V⊆A›‹A⊆⋃T›
        by blast
      then have ((cl(V)∩(Closure(W,(T{restricted to}cl(V))))))=Closure(W,(T{restricted
to}cl(V))) by auto
      ultimately have Closure(W,(T{restricted to}cl(V))){is compact in}(T{restricted
to}cl(V)) by auto
      then have Closure(W,(T{restricted to}cl(V))){is compact in}T us-
ing compact_subspace_imp_compact[of Closure(W,T{restricted to}cl(V))]
        cont by auto
      with A1 A2 have ∃c∈Pow(U). x∈int(c)∧c{is compact in}T by auto
    }
    then have ∀U∈T. x∈U ⟶ (∃c∈Pow(U). x∈int(c)∧c{is compact in}T)
by auto
  }
  then show thesis unfolding IsLocally_def[OF topSpaceAssum] IsLocallyComp_def
by auto
qed
```

88.4 Compactification by one point

Given a topological space, we can always add one point to the space and get a new compact topology; as we will check in this section.

```
definition
  OPCompactification ({one-point compactification of}_ 90)
  where {one-point compactification of}T≡T∪{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T).
B{is compact in}T ∧ B{is closed in}T}}
```

Firstly, we check that what we defined is indeed a topology.

```
theorem (in topology0) op_comp_is_top:
  shows ({one-point compactification of}T){is a topology} unfolding IsATopology_def
proof(safe)
  fix M assume M⊆{one-point compactification of}T
  then have disj:M⊆T∪{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact in}T
∧ B{is closed in}T}} unfolding OPCompactification_def by auto
  let MT={A∈M. A∈T}
  have MT⊆T by auto
```

```
  then have c1:⋃MT∈T using topSpaceAssum unfolding IsATopology_def by
auto
  let MK={A∈M. A∉T}
  have ⋃M=⋃MK ∪ ⋃MT by auto
  from disj have MK⊆{A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact
in}T ∧ B{is closed in}T}}} by auto
  moreover have N:⋃T∉(⋃T) using mem_not_refl by auto
  {
    fix B assume B∈M B∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact
in}T ∧ B{is closed in}T}}
    then obtain K where K∈Pow(⋃T) B={⋃T}∪((⋃T)-K) by auto
    with N have ⋃T∈B by auto
    with N have B∉T by auto
    with ‹B∈M› have B∈MK by auto
  }
  then have {A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact in}T
∧ B{is closed in}T}}}⊆MK by auto
  ultimately have MK_def:MK={A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T).
B{is compact in}T ∧ B{is closed in}T}}} by auto
  let KK={K∈Pow(⋃T). {⋃T}∪((⋃T)-K)∈MK}
  {
    assume MK=0
    then have ⋃M=⋃MT by auto
    then have ⋃M∈T using c1 by auto
    then have ⋃M∈{one-point compactification of}T unfolding OPCompactification_def
by auto
  }
  moreover
  {
    assume MK≠0
    then obtain A where A∈MK by auto
    then obtain K1 where A={⋃T}∪((⋃T)-K1) K1∈Pow(⋃T) K1{is closed
in}T K1{is compact in}T using MK_def by auto
    with ‹A∈MK› have ⋂KK⊆K1 by auto
    from ‹A∈MK› ‹A={⋃T}∪((⋃T)-K1)› ‹K1∈Pow(⋃T)› have KK≠0 by blast
    {
      fix K assume K∈KK
      then have {⋃T}∪((⋃T)-K)∈MK K⊆⋃T by auto
      then obtain KK where A:{⋃T}∪((⋃T)-K)={⋃T}∪((⋃T)-KK) KK⊆⋃T
KK{is compact in}T KK{is closed in}T using MK_def by auto
      note A(1) moreover
      have (⋃T)-K⊆{⋃T}∪((⋃T)-K) (⋃T)-KK⊆{⋃T}∪((⋃T)-KK) by auto
      ultimately have (⋃T)-K⊆{⋃T}∪((⋃T)-KK) (⋃T)-KK⊆{⋃T}∪((⋃T)-K)
by auto moreover
      from N have ⋃T∉(⋃T)-K ⋃T∉(⋃T)-KK by auto ultimately
      have (⋃T)-K⊆((⋃T)-KK) (⋃T)-KK⊆((⋃T)-K) by auto
      then have (⋃T)-K=(⋃T)-KK by auto moreover
      from ‹K⊆⋃T› have K=(⋃T)-((⋃T)-K) by auto ultimately
      have K=(⋃T)-((⋃T)-KK) by auto
```

```
      with ‹KK⊆⋃T› have K=KK by auto
      with A(4) have K{is closed in}T by auto
    }
    then have ∀K∈KK. K{is closed in}T by auto
    with ‹KK≠0› have (⋂KK){is closed in}T using Top_3_L4 by auto
    with ‹K1{is compact in}T› have (K1∩(⋂KK)){is compact in}T using
Compact_is_card_nat
      compact_closed[of K1natT⋂KK] by auto moreover
    from ‹⋂KK⊆K1› have K1∩(⋂KK)=(⋂KK) by auto ultimately
    have (⋂KK){is compact in}T by auto
    with ‹(⋂KK){is closed in}T› ‹⋂KK⊆K1› ‹K1∈Pow(⋃T)› have ({⋃T}∪((⋃T)-(⋂KK)))∈({one
compactification of}T)
      unfolding OPCompactification_def by blast
    have t:⋃MK=⋃{A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact
in}T ∧ B{is closed in}T}}}
      using MK_def by auto
    {
      fix x assume x∈⋃MK
      with t have x∈⋃{A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is
compact in}T ∧ B{is closed in}T}}} by auto
      then have ∃AA∈{A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact
in}T ∧ B{is closed in}T}}}. x∈AA
        using Union_iff by auto
      then obtain AA where AAp:AA∈{A∈M. A∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T).
B{is compact in}T ∧ B{is closed in}T}}} x∈AA by auto
      then obtain K2 where AA={⋃T}∪((⋃T)-K2) K2∈Pow(⋃T)K2{is compact
in}T K2{is closed in}T by auto
      with ‹x∈AA› have x=⋃T ∨ (x∈(⋃T) ∧ x∉K2) by auto
      from ‹K2∈Pow(⋃T)› ‹AA={⋃T}∪((⋃T)-K2)› AAp(1) MK_def have K2∈KK
by auto
      then have ⋂KK⊆K2 by auto
      with ‹x=⋃T ∨ (x∈(⋃T) ∧ x∉K2)› have x=⋃T∨(x∈⋃T ∧ x∉⋂KK) by
auto
     then have x∈{⋃T}∪((⋃T)-(⋂KK)) by auto
    }
    then have ⋃MK⊆{⋃T}∪((⋃T)-(⋂KK)) by auto
    moreover
    {
      fix x assume x∈{⋃T}∪((⋃T)-(⋂KK))
      then have x=⋃T∨(x∈(⋃T)∧ x∉⋂KK) by auto
      with ‹KK≠0› obtain K2 where K2∈KK x=⋃T∨(x∈⋃T∧ x∉K2) by auto
      then have {⋃T}∪((⋃T)-K2)∈MK by auto
      with ‹x=⋃T∨(x∈⋃T∧ x∉K2)› have x∈⋃MK by auto
    }
    then have {⋃T}∪((⋃T)-(⋂KK))⊆⋃MK by (safe,auto)
    ultimately have ⋃MK={⋃T}∪((⋃T)-(⋂KK)) by blast
    from ‹⋃MT∈T› have ⋃T-(⋃T-⋃MT)=⋃MT by auto
    with ‹⋃MT∈T› have (⋃T-⋃MT){is closed in}T unfolding IsClosed_def
by auto
```

```
    have ((⋃T)-(⋂KK))∪(⋃T-(⋃T-⋃MT))=(⋃T)-((⋂KK)∩(⋃T-⋃MT)) by
auto
    then have ({⋃T}∪((⋃T)-(⋂KK)))∪(⋃T-(⋃T-⋃MT))={⋃T}∪((⋃T)-((⋂KK)∩(⋃T-⋃MT)))
by auto
    with ‹⋃MK={⋃T}∪((⋃T)-(⋂KK))›‹⋃T-(⋃T-⋃MT)=⋃MT› have ⋃MK∪⋃MT={⋃T}∪((⋃T)-((⋂K
    by auto
    with ‹⋃M=⋃MK ∪⋃MT› have unM:⋃M={⋃T}∪((⋃T)-((⋂KK)∩(⋃T-⋃MT)))
by auto
    have ((⋂KK)∩(⋃T-⋃MT)) {is closed in}T using ‹(⋂KK){is closed in}T›‹(⋃T-(⋃MT)){is
closed in}T›
      Top_3_L5 by auto
    moreover
    note ‹(⋃T-(⋃MT)){is closed in}T› ‹(⋂KK){is compact in}T›
    then have ((⋂KK)∩(⋃T-⋃MT)){is compact of cardinal}nat{in}T us-
ing compact_closed[of ⋂KKnatT(⋃T-⋃MT)] Compact_is_card_nat
      by auto
    then have ((⋂KK)∩(⋃T-⋃MT)){is compact in}T using Compact_is_card_nat
by auto
    ultimately have {⋃T}∪(⋃T-((⋂KK)∩(⋃T-⋃MT)))∈{one-point compactification
of}T
      unfolding OPCompactification_def IsClosed_def by auto
    with unM have ⋃M∈{one-point compactification of}T by auto
  }
  ultimately show ⋃M∈{one-point compactification of}T by auto
next
  fix U V assume U∈{one-point compactification of}T and V∈{one-point
compactification of}T
  then have A:U∈T∨(∃KU∈Pow(⋃T). U={⋃T}∪(⋃T-KU)∧KU{is closed in}T∧KU{is
compact in}T)
    V∈T∨(∃KV∈Pow(⋃T). V={⋃T}∪(⋃T-KV)∧KV{is closed in}T∧KV{is compact
in}T) unfolding OPCompactification_def
    by auto
 have N:⋃T∉(⋃T) using mem_not_refl by auto
  {
    assume U∈TV∈T
    then have U∩V∈T using topSpaceAssum unfolding IsATopology_def by
auto
    then have U∩V∈{one-point compactification of}T unfolding OPCompactification_def
    by auto
  }
  moreover
  {
    assume U∈TV∉T
    then obtain KV where V:KV{is closed in}TKV{is compact in}TV={⋃T}∪(⋃T-KV)
    using A(2) by auto
    with N ‹U∈T› have ⋃T∉U by auto
    then have ⋃T∉U∩V by auto
    then have U∩V=U∩(⋃T-KV) using V(3) by auto
    moreover have ⋃T-KV∈T using V(1) unfolding IsClosed_def by auto
```

```
    with ‹U∈T› have U∩(⋃T-KV)∈T using topSpaceAssum unfolding IsATopology_def
by auto
    with ‹U∩V=U∩(⋃T-KV)› have U∩V∈T by auto
    then have U∩V∈{one-point compactification of}T unfolding OPCompactification_def
by auto
    }
  moreover
  {
    assume U∉TV∈T
    then obtain KV where V:KV{is closed in}TKV{is compact in}TU={⋃T}∪(⋃T-KV)
    using A(1) by auto
    with N ‹V∈T› have ⋃T∉V by auto
    then have ⋃T∉U∩V by auto
    then have U∩V=(⋃T-KV)∩V using V(3) by auto
    moreover have ⋃T-KV∈T using V(1) unfolding IsClosed_def by auto
    with ‹V∈T› have (⋃T-KV)∩V∈T using topSpaceAssum unfolding IsATopology_def
by auto
    with ‹U∩V=(⋃T-KV)∩V› have U∩V∈T by auto
    then have U∩V∈{one-point compactification of}T unfolding OPCompactification_def
by auto
  }
  moreover
  {
    assume U∉TV∉T
    then obtain KV KU where V:KV{is closed in}TKV{is compact in}TV={⋃T}∪(⋃T-KV)
     and U:KU{is closed in}TKU{is compact in}TU={⋃T}∪(⋃T-KU)
    using A by auto
    with V(3) U(3) have ⋃T∈U∩V by auto
    then have U∩V={⋃T}∪((⋃T-KV)∩(⋃T-KU)) using V(3) U(3) by auto
    moreover have ⋃T-KV∈T⋃T-KU∈T using V(1) U(1) unfolding IsClosed_def
by auto
    then have (⋃T-KV)∩(⋃T-KU)∈T using topSpaceAssum unfolding IsATopology_def
by auto
    then have (⋃T-KV)∩(⋃T-KU)=⋃T-(⋃T-((⋃T-KV)∩(⋃T-KU))) by auto
moreover
    with ‹(⋃T-KV)∩(⋃T-KU)∈T› have (⋃T-(⋃T-KV)∩(⋃T-KU)){is closed
in}T unfolding IsClosed_def
      by auto moreover
    from V(1) U(1) have (⋃T-(⋃T-KV)∩(⋃T-KU))=KV∪KU unfolding IsClosed_def
by auto
    with V(2) U(2) have (⋃T-(⋃T-KV)∩(⋃T-KU)){is compact in}T using
union_compact[of KVnatTKU] Compact_is_card_nat
      InfCard_nat by auto ultimately
    have U∩V∈{one-point compactification of}T unfolding OPCompactification_def
by auto
  }
  ultimately show U∩V∈{one-point compactification of}T by auto
qed
```

The original topology is an open subspace of the new topology.

```
theorem (in topology0) open_subspace:
  shows ⋃T∈{one-point compactification of}T and ({one-point compactification
of}T){restricted to}⋃T=T
proof-
  show ⋃T∈{one-point compactification of}T
  unfolding OPCompactification_def using topSpaceAssum unfolding IsATopology_def
by auto
  have T⊆({one-point compactification of}T){restricted to}⋃T unfold-
ing OPCompactification_def RestrictedTo_def by auto
  moreover
  {
    fix A assume A∈({one-point compactification of}T){restricted to}⋃T
    then obtain R where R∈({one-point compactification of}T) A=⋃T∩R
unfolding RestrictedTo_def by auto
    then obtain K where K:R∈T ∨ (R={⋃T}∪(⋃T-K) ∧ K{is closed in}T)
unfolding OPCompactification_def by auto
    with ‹A=⋃T∩R› have (A=R∧R∈T)∨(A=⋃T-K ∧ K{is closed in}T) using
mem_not_refl unfolding IsClosed_def by auto
    with K have A∈T unfolding IsClosed_def by auto
  }
  ultimately
  show ({one-point compactification of}T){restricted to}⋃T=T by auto
qed
```

We added only one new point to the space.

```
lemma (in topology0) op_compact_total:
  shows ⋃({one-point compactification of}T)={⋃T}∪(⋃T)
proof-
  have 0{is compact in}T unfolding IsCompact_def FinPow_def by auto
  moreover note Top_3_L2 ultimately have TT:0∈{A∈Pow(⋃T). A{is compact
in}T ∧A{is closed in}T} by auto
  have ⋃({one-point compactification of}T)=(⋃T)∪(⋃{{⋃T}∪(⋃T-K). K∈{B∈Pow(⋃T).
B{is compact in}T∧B{is closed in}T}}) unfolding OPCompactification_def
    by blast
  also have ...=(⋃T)∪{⋃T}∪(⋃{(⋃T-K). K∈{B∈Pow(⋃T). B{is compact in}T∧B{is
closed in}T}}) using TT by auto
  ultimately show ⋃({one-point compactification of}T)={⋃T}∪(⋃T) by
auto
qed
```

The one point compactification, gives indeed a compact topological space.

```
theorem (in topology0) compact_op:
  shows ({⋃T}∪(⋃T)){is compact in}({one-point compactification of}T)
unfolding IsCompact_def
proof(safe)
  have 0{is compact in}T unfolding IsCompact_def FinPow_def by auto
  moreover note Top_3_L2 ultimately have 0∈{A∈Pow(⋃T). A{is compact
in}T ∧A{is closed in}T} by auto
  then have {⋃T}∪(⋃T)∈{one-point compactification of}T unfolding OPCompactification_def
```

```
by auto
  then show ⋃T ∈ ⋃{one-point compactification of}T by auto
next
  fix x B assume x∈BB∈T
  then show x∈⋃({one-point compactification of}T) using open_subspace
by auto
next
  fix M assume A:M⊆({one-point compactification of}T) {⋃T} ∪ ⋃T ⊆ ⋃M
  then obtain R where R∈M⋃T∈R by auto
  have ⋃T∉⋃T using mem_not_refl by auto
  with ‹R∈M› ‹⋃T∈R› A(1) obtain K where K:R={⋃T}∪(⋃T-K) K{is compact
in}TK{is closed in}T
    unfolding OPCompactification_def by auto
  from K(1,2) have B:{⋃T} ∪ (⋃T) = R ∪ K unfolding IsCompact_def by
auto
  with A(2) have K⊆⋃M by auto
  from K(2) have K{is compact in}(({one-point compactification of}T){restricted
to}⋃T) using open_subspace(2)
    by auto
  then have K{is compact in}({one-point compactification of}T) using
compact_subspace_imp_compact
    ‹K{is closed in}T› unfolding IsClosed_def by auto
  with ‹K⊆⋃M› A(1) have (∃N∈FinPow(M). K ⊆ ⋃N) unfolding IsCompact_def
by auto
  then obtain N where N∈FinPow(M) K⊆⋃N by auto
  with ‹R∈M› have (N ∪{R})∈FinPow(M)R∪K⊆⋃(N∪{R}) unfolding FinPow_def
by auto
  with B show ∃N∈FinPow(M). {⋃T} ∪ (⋃T)⊆⋃N by auto
qed
```

The one point compactification is Hausdorff iff the original space is also Hausdorff and locally compact.

```
lemma (in topology0) op_compact_T2_1:
  assumes ({one-point compactification of}T){is T2}
  shows T{is T2}
  using T2_here[OF assms, of ⋃T] open_subspace by auto

lemma (in topology0) op_compact_T2_2:
  assumes ({one-point compactification of}T){is T2}
  shows T{is locally-compact}
proof-
  {
    fix x assume x∈⋃T
    then have x∈{⋃T}∪(⋃T) by auto
    moreover have ⋃T∈{⋃T}∪(⋃T) by auto moreover
    from ‹x∈⋃T› have x≠⋃T using mem_not_refl by auto
    ultimately have ∃U∈{one-point compactification of}T. ∃V∈{one-point
compactification of}T. x ∈ U ∧ (⋃T) ∈ V ∧ U ∩ V = 0
      using assms op_compact_total unfolding isT2_def by auto
```

```
    then obtain U V where UV:U∈{one-point compactification of}TV∈{one-point
compactification of}T
      x∈U⋃T∈VU∩V=0 by auto
    from ‹V∈{one-point compactification of}T› ‹⋃T∈V› mem_not_refl ob-
tain K where K:V={⋃T}∪(⋃T-K)K{is closed in}TK{is compact in}T
      unfolding OPCompactification_def by auto
    from ‹U∈{one-point compactification of}T› have U⊆{⋃T}∪(⋃T) un-
folding OPCompactification_def
      using op_compact_total by auto
    with ‹U∩V=0› K have U⊆KK⊆⋃T unfolding IsClosed_def by auto
    then have (⋃T)∩U=U by auto moreover
    from UV(1) have ((⋃T)∩U)∈({one-point compactification of}T){restricted
to}⋃T
      unfolding RestrictedTo_def by auto
    ultimately have U∈T using open_subspace(2) by auto
    with ‹x∈U›‹U⊆K› have x∈int(K) using Top_2_L6 by auto
    with ‹K⊆⋃T› ‹K{is compact in}T› have ∃A∈Pow(⋃T). x∈int(A)∧ A{is
compact in}T by auto
  }
  then have ∀x∈⋃T. ∃A∈Pow(⋃T). x∈int(A)∧ A{is compact in}T by auto
  then show thesis using op_compact_T2_1[OF assms] exist_compact_neig_T2_imp_locally_compa
    by auto
qed

lemma (in topology0) op_compact_T2_3:
  assumes T{is locally-compact} T{is T₂}
  shows ({one-point compactification of}T){is T₂}
proof-
  {
    fix x y assume x≠yx∈⋃({one-point compactification of}T)y∈⋃({one-point
compactification of}T)
    then have S:x∈{⋃T}∪(⋃T)y∈{⋃T}∪(⋃T) using op_compact_total by
auto
    {
      assume x∈⋃Ty∈⋃T
      with ‹x≠y› have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 using assms(2) un-
folding isT2_def by auto
      then have ∃U∈({one-point compactification of}T). ∃V∈({one-point
compactification of}T). x∈U∧y∈V∧U∩V=0
        unfolding OPCompactification_def by auto
    }
    moreover
    {
      assume x∉⋃T∨y∉⋃T
      with S have x=⋃T∨y=⋃T by auto
      with ‹x≠y› have (x=⋃T∧y≠⋃T)∨(y=⋃T∧x≠⋃T) by auto
      with S have (x=⋃T∧y∈⋃T)∨(y=⋃T∧x∈⋃T) by auto
      then obtain Ky Kx where (x=⋃T∧ Ky{is compact in}T∧y∈int(Ky))∨(y=⋃T∧
Kx{is compact in}T∧x∈int(Kx))
```

```
        using assms(1) locally_compact_exist_compact_neig by blast
      then have (x=⋃T∧ Ky{is compact in}T∧ Ky{is closed in}T∧y∈int(Ky))∨(y=⋃T∧
Kx{is compact in}T∧ Kx{is closed in}T∧x∈int(Kx))
        using in_t2_compact_is_cl assms(2) by auto
      then have (x∈{⋃T}∪(⋃T-Ky)∧y∈int(Ky)∧ Ky{is compact in}T∧ Ky{is
closed in}T)∨(y∈{⋃T}∪(⋃T-Kx)∧x∈int(Kx)∧ Kx{is compact in}T∧ Kx{is
closed in}T)
        by auto moreover
      {
        fix K
        assume A:K{is closed in}TK{is compact in}T
        then have K⊆⋃T unfolding IsClosed_def by auto
        moreover have ⋃T∉⋃T using mem_not_refl by auto
        ultimately have ({⋃T}∪(⋃T-K))∩K=0 by auto
        then have ({⋃T}∪(⋃T-K))∩int(K)=0 using Top_2_L1 by auto more-
over
        from A have {⋃T}∪(⋃T-K)∈({one-point compactification of}T)
unfolding OPCompactification_def
          IsClosed_def by auto moreover
        have int(K)∈({one-point compactification of}T) using Top_2_L2
unfolding OPCompactification_def
          by auto ultimately
        have int(K)∈({one-point compactification of}T)∧{⋃T}∪(⋃T-K)∈({one-point
compactification of}T)∧({⋃T}∪(⋃T-K))∩int(K)=0
          by auto
      }
      ultimately have ({⋃T} ∪ (⋃T - Ky)∈({one-point compactification
of}T)∧int(Ky)∈({one-point compactification of}T)∧x ∈ {⋃T} ∪ (⋃T - Ky)
∧ y ∈ int(Ky) ∧ ({⋃T}∪(⋃T-Ky))∩int(Ky)=0) ∨
        ({⋃T} ∪ (⋃T - Kx)∈({one-point compactification of}T)∧int(Kx)∈({one-point
compactification of}T)∧y ∈ {⋃T} ∪ (⋃T - Kx) ∧ x ∈ int(Kx) ∧ ({⋃T}∪(⋃T-Kx))∩int(Kx)=0)
by auto
      moreover
      {
        assume ({⋃T} ∪ (⋃T - Ky)∈({one-point compactification of}T)∧int(Ky)∈({one-point
compactification of}T)∧x ∈ {⋃T} ∪ (⋃T - Ky) ∧ y ∈ int(Ky) ∧ ({⋃T}∪(⋃T-Ky))∩int(Ky)=0)
        then have ∃U∈({one-point compactification of}T). ∃V∈({one-point
compactification of}T). x∈U∧y∈V∧U∩V=0 using exI[OF exI[of _ int(Ky)],of
λU V. U∈({one-point compactification of}T)∧V∈({one-point compactification
of}T) ∧ x∈U∧y∈V∧U∩V=0 {⋃T}∪(⋃T-Ky)]
          by auto
      } moreover
      {
        assume ({⋃T} ∪ (⋃T - Kx)∈({one-point compactification of}T)∧int(Kx)∈({one-point
compactification of}T)∧y ∈ {⋃T} ∪ (⋃T - Kx) ∧ x ∈ int(Kx) ∧ ({⋃T}∪(⋃T-Kx))∩int(Kx)=0)
        then have ∃U∈({one-point compactification of}T). ∃V∈({one-point
compactification of}T). x∈U∧y∈V∧U∩V=0 using exI[OF exI[of _ {⋃T}∪(⋃T-Kx)],of
λU V. U∈({one-point compactification of}T)∧V∈({one-point compactification
of}T) ∧ x∈U∧y∈V∧U∩V=0int(Kx) ]
```

```
        by blast
      }
      ultimately have ∃U∈({one-point compactification of}T). ∃V∈({one-point
compactification of}T). x∈U∧y∈V∧U∩V=0 by auto
    }
    ultimately have ∃U∈({one-point compactification of}T). ∃V∈({one-point
compactification of}T). x∈U∧y∈V∧U∩V=0 by auto
  }
  then show thesis unfolding isT2_def by auto
qed
```

In conclusion, every locally compact Hausdorff topological space is regular; since this property is hereditary.

```
corollary (in topology0) locally_compact_T2_imp_regular:
  assumes T{is locally-compact} T{is T₂}
  shows T{is regular}
proof-
  from assms have ( {one-point compactification of}T) {is T₂} using op_compact_T2_3
by auto
  then have ({one-point compactification of}T) {is T₄} unfolding isT4_def
using T2_is_T1 topology0.T2_compact_is_normal
    op_comp_is_top unfolding topology0_def using op_compact_total compact_op
by auto
  then have ({one-point compactification of}T) {is T₃} using topology0.T4_is_T3
op_comp_is_top unfolding topology0_def
    by auto
  then have ({one-point compactification of}T) {is regular} using isT3_def
by auto moreover
  have ⋃T⊆⋃({one-point compactification of}T) using op_compact_total
by auto
  ultimately have (({one-point compactification of}T){restricted to}⋃T)
{is regular} using regular_here by auto
  then show T{is regular} using open_subspace(2) by auto
qed
```

This last corollary has an explanation: In Hausdorff spaces, compact sets are closed and regular spaces are exactly the "locally closed spaces"(those which have a neighbourhood basis of closed sets). So the neighbourhood basis of compact sets also works as the neighbourhood basis of closed sets we needed to find.

```
definition
  IsLocallyClosed (_{is locally-closed})
  where T{is locally-closed} ≡ T{is locally}(λB TT. B{is closed in}TT)

lemma (in topology0) regular_locally_closed:
  shows T{is regular} ⟷ (T{is locally-closed})
proof
  assume T{is regular}
```

```
  then have a:∀x∈⋃T. ∀U∈T. (x∈U) ⟶ (∃V∈T. x ∈ V ∧ cl(V) ⊆ U) using regular_imp_exist_clos_neig by auto
  {
    fix x b assume x∈⋃Tb∈Tx∈b
    with a obtain V where V∈Tx∈Vcl(V)⊆b by blast
    note ‹cl(V)⊆b› moreover
    from ‹V∈T› have V⊆⋃T by auto
    then have V⊆cl(V) using cl_contains_set by auto
    with ‹x∈V›‹V∈T› have x∈int(cl(V)) using Top_2_L6 by auto moreover
    from ‹V⊆⋃T› have cl(V){is closed in}T using cl_is_closed by auto
    ultimately have x∈int(cl(V))cl(V)⊆bcl(V){is closed in}T by auto
    then have ∃K∈Pow(b). x∈int(K)∧K{is closed in}T by auto
  }
  then show T{is locally-closed} unfolding IsLocally_def[OF topSpaceAssum]
IsLocallyClosed_def
    by auto
next
  assume T{is locally-closed}
  then have a:∀x∈⋃T. ∀b∈T. x∈b ⟶ (∃K∈Pow(b). x∈int(K)∧K{is closed
in}T) unfolding IsLocally_def[OF topSpaceAssum]
    IsLocallyClosed_def by auto
  {
    fix x b assume x∈⋃Tb∈Tx∈b
    with a obtain K where K:K⊆bx∈int(K)K{is closed in}T by blast
    have int(K)⊆K using Top_2_L1 by auto
    with K(3) have cl(int(K))⊆K using Top_3_L13 by auto
    with K(1) have cl(int(K))⊆b by auto moreover
    have int(K)∈T using Top_2_L2 by auto moreover
    note ‹x∈int(K)› ultimately have ∃V∈T. x∈V∧ cl(V)⊆b by auto
  }
  then have ∀x∈⋃T. ∀b∈T. x∈b ⟶ (∃V∈T. x∈V∧ cl(V)⊆b) by auto
  then show T{is regular} using exist_clos_neig_imp_regular by auto
qed
```

88.5 Hereditary properties and local properties

In this section, we prove a relation between a property and its local property for hereditary properties. Then we apply it to locally-Hausdorff or locally-T_2. We also prove the relation between locally-T_2 and another property that appeared when considering anti-properties, the anti-hyperconnectness.

If a property is hereditary in open sets, then local properties are equivalent to find just one open neighbourhood with that property instead of a whole local basis.

```
lemma (in topology0) her_P_is_loc_P:
  assumes ∀TT. ∀B∈Pow(⋃TT). ∀A∈TT. TT{is a topology}∧P(B,TT) ⟶ P(B∩A,TT)
  shows (T{is locally}P) ⟷ (∀x∈⋃T. ∃A∈T. x∈A∧P(A,T))
proof
```

```
  assume A:T{is locally}P
  {
    fix x assume x:x∈⋃T
    with A have ∀b∈T. x∈b ⟶ (∃c∈Pow(b). x∈int(c)∧P(c,T)) unfolding
IsLocally_def[OF topSpaceAssum]
      by auto moreover
    note x moreover
    have ⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto
    ultimately have ∃c∈Pow(⋃T). x∈int(c)∧ P(c,T) by auto
    then obtain c where c:c⊆⋃Tx∈int(c)P(c,T) by auto
    have P:int(c)∈T using Top_2_L2 by auto moreover
    from c(1,3) topSpaceAssum assms have ∀A∈T. P(c∩A,T) by auto
    ultimately have P(c∩int(c),T) by auto moreover
    from Top_2_L1[of c] have int(c)⊆c by auto
    then have c∩int(c)=int(c) by auto
    ultimately have P(int(c),T) by auto
    with P c(2) have ∃V∈T. x∈V∧P(V,T) by auto
  }
  then show ∀x∈⋃T. ∃V∈T. x∈V∧P(V,T) by auto
  next
  assume A:∀x∈⋃T. ∃A∈T. x ∈ A ∧ P(A, T)
  {
    fix x assume x:x∈⋃T
    {
      fix b assume b:x∈bb∈T
      from x A obtain A where A_def:A∈Tx∈AP(A,T) by auto
      from A_def(1,3) assms topSpaceAssum have ∀G∈T. P(A∩G,T) by auto
      with b(2) have P(A∩b,T) by auto
      moreover from b(1) A_def(2) have x∈A∩b by auto moreover
      have A∩b∈T using b(2) A_def(1) topSpaceAssum IsATopology_def by
auto
      then have int(A∩b)=A∩b using Top_2_L3 by auto
      ultimately have x∈int(A∩b)∧P(A∩b,T) by auto
      then have ∃c∈Pow(b). x∈int(c)∧P(c,T) by auto
    }
    then have ∀b∈T. x∈b⟶(∃c∈Pow(b). x∈int(c)∧P(c,T)) by auto
  }
  then show T{is locally}P unfolding IsLocally_def[OF topSpaceAssum]
by auto
qed


definition
  IsLocallyT2 (_{is locally-T₂} 70)
  where T{is locally-T₂}≡T{is locally}(λB. λT. (T{restricted to}B){is
T₂})
```

Since T_2 is an hereditary property, we can apply the previous lemma.

```
corollary (in topology0) loc_T2:
```

```
  shows (T{is locally-T_2}) ⟷ (∀x∈⋃T. ∃A∈T. x∈A∧(T{restricted to}A){is
T_2})
proof-
  {
    fix TT B A assume TT:TT{is a topology} (TT{restricted to}B){is T_2}
A∈TTB∈Pow(⋃TT)
    then have s:B∩A⊆BB⊆⋃TT by auto
    then have (TT{restricted to}(B∩A))=(TT{restricted to}B){restricted
to}(B∩A) using subspace_of_subspace
      by auto moreover
    have ⋃(TT{restricted to}B)=B unfolding RestrictedTo_def using s(2)
by auto
    then have B∩A⊆⋃(TT{restricted to}B) using s(1) by auto moreover
    note TT(2) ultimately have (TT{restricted to}(B∩A)){is T_2} using T2_here
      by auto
  }
  then have ∀TT. ∀B∈Pow(⋃TT). ∀A∈TT. TT{is a topology}∧(TT{restricted
to}B){is T_2} ⟶ (TT{restricted to}(B∩A)){is T_2}
    by auto
  with her_P_is_loc_P[where P=λA. λTT. (TT{restricted to}A){is T_2}] show
thesis unfolding IsLocallyT2_def by auto
qed
```

First, we prove that a locally-T_2 space is anti-hyperconnected.

Before starting, let's prove that an open subspace of an hyperconnected space is hyperconnected.

```
lemma(in topology0) open_subspace_hyperconn:
  assumes T{is hyperconnected} U∈T
  shows (T{restricted to}U){is hyperconnected}
proof-
  {
    fix A B assume A∈(T{restricted to}U)B∈(T{restricted to}U)A∩B=0
    then obtain AU BU where A=U∩AUB=U∩BU AU∈TBU∈T unfolding RestrictedTo_def
by auto
    then have A∈TB∈T using topSpaceAssum assms(2) unfolding IsATopology_def
by auto
    with ‹A∩B=0› have A=0∨B=0 using assms(1) unfolding IsHConnected_def
by auto
  }
  then show thesis unfolding IsHConnected_def by auto
qed

lemma(in topology0) locally_T2_is_antiHConn:
  assumes T{is locally-T_2}
  shows T{is anti-}IsHConnected
proof-
  {
    fix A assume A:A∈Pow(⋃T)(T{restricted to}A){is hyperconnected}
```

{

fix x assume x∈A

with A(1) have x∈⋃T by auto moreover

have ⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto ultimately

have ∃c∈Pow(⋃T). x ∈ int(c) ∧ (T {restricted to} c) {is T_2} using assms

unfolding IsLocallyT2_def IsLocally_def[OF topSpaceAssum] by auto

then obtain c where c:c∈Pow(⋃T)x∈int(c)(T {restricted to} c) {is T_2} by auto

have ⋃(T {restricted to} c)=(⋃T)∩c unfolding RestrictedTo_def by auto

with ‹c∈Pow(⋃T)›‹⋃T∈T› have tot:⋃(T {restricted to} c)=c by auto

have int(c)∈T using Top_2_L2 by auto

then have A∩(int(c))∈(T{restricted to}A) unfolding RestrictedTo_def by auto

with A(2) have ((T{restricted to}A){restricted to}(A∩(int(c)))){is hyperconnected}

using topology0.open_subspace_hyperconn unfolding topology0_def using Top_1_L4

by auto

then have (T{restricted to}(A∩(int(c)))){is hyperconnected} using subspace_of_subspace[of A∩(int(c))

AT] A(1) by force moreover

have int(c)⊆c using Top_2_L1 by auto

then have sub:A∩(int(c))⊆c by auto

then have A∩(int(c))⊆⋃(T {restricted to} c) using tot by auto

then have ((T {restricted to} c){restricted to}(A∩(int(c)))) {is T_2} using

T2_here[OF c(3)] by auto

with sub have (T {restricted to}(A∩(int(c)))){is T_2} using subspace_of_subspace[of A∩(int(c))

cT] ‹c∈Pow(⋃T)› by auto

ultimately have (T{restricted to}(A∩(int(c)))){is hyperconnected}(T {restricted to}(A∩(int(c)))){is T_2}

by auto

then have (T{restricted to}(A∩(int(c)))){is hyperconnected}(T {restricted to}(A∩(int(c)))){is anti-}IsHConnected

using topology0.T2_imp_anti_HConn unfolding topology0_def using Top_1_L4 by auto

moreover

have ⋃(T{restricted to}(A∩(int(c))))=(⋃T)∩A∩(int(c)) unfolding RestrictedTo_def by auto

with A(1) Top_2_L2 have ⋃(T{restricted to}(A∩(int(c))))=A∩(int(c)) by auto

then have A∩(int(c))⊆⋃(T{restricted to}(A∩(int(c)))) by auto moreover

have A∩(int(c))⊆⋃T using A(1) Top_2_L2 by auto

```
      then have (T{restricted to}(A∩(int(c)))){restricted to}(A∩(int(c)))=(T{restricted
to}(A∩(int(c))))
        using subspace_of_subspace[of A∩(int(c))A∩(int(c))T] by auto
      ultimately have (A∩(int(c))){is in the spectrum of}IsHConnected
unfolding antiProperty_def
        by auto
      then have A∩(int(c))≲1 using HConn_spectrum by auto
      then have (A∩(int(c))={x}) using lepoll_1_is_sing ‹x∈A›‹x∈int(c)›
by auto
      then have {x}∈(T{restricted to}A) using ‹(A∩(int(c))∈(T{restricted
to}A))› by auto
    }
    then have pointOpen:∀x∈A. {x}∈(T{restricted to}A) by auto
    {
      fix x y assume x≠yx∈Ay∈A
      with pointOpen have {x}∈(T{restricted to}A){y}∈(T{restricted to}A){x}∩{y}=0
        by auto
      with A(2) have {x}=0∨{y}=0 unfolding IsHConnected_def by auto
      then have False by auto
    }
    then have uni:∀x∈A. ∀y∈A. x=y by auto
    {
      assume A≠0
      then obtain x where x∈A by auto
      with uni have A={x} by auto
      then have A≈1 using singleton_eqpoll_1 by auto
      then have A≲1 using eqpoll_imp_lepoll by auto
    }
    moreover
    {
      assume A=0
      then have A≈0 by auto
      then have A≲1 using empty_lepollI eq_lepoll_trans by auto
    }
    ultimately have A≲1 by auto
    then have A{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
  }
  then show thesis unfolding antiProperty_def by auto
qed
```

Now we find a counter-example for: Every anti-hyperconnected space is locally-Hausdorff.

The example we are going to consider is the following. Put in X an anti-hyperconnected topology, where an infinite number of points don't have finite sets as neighbourhoods. Then add a new point to the set, $p \notin X$. Consider the open sets on $X \cup p$ as the anti-hyperconnected topology and the open sets that contain p are $p \cup A$ where $X \setminus A$ is finite.

This construction equals the one-point compactification iff X is anti-compact; i.e., the only compact sets are the finite ones. In general this topology is contained in the one-point compactification topology, making it compact too.

It is easy to check that any open set containing p meets infinite other non-empty open set. The question is if such a topology exists.

```
theorem (in topology0) COF_comp_is_top:
  assumes T{is T₁}¬(⋃T≺nat)
  shows ((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)
{is a topology}
proof-
  have N:⋃T∉(⋃T) using mem_not_refl by auto
  {
    fix M assume M:M⊆((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)
    let MT={A∈M. A∈T}
    let MK={A∈M. A∉T}
    have MM:(⋃MT)∪(⋃MK)=⋃M by auto
    have MN:⋃MT∈T using topSpaceAssum unfolding IsATopology_def by auto
    then have sub:MK⊆({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
      using M by auto
    then have MK⊆({one-point compactification of}(CoFinite (⋃T))) by
auto
    then have CO:⋃MK∈({one-point compactification of}(CoFinite (⋃T)))
using
      topology0.op_comp_is_top[OF topology0_CoCardinal[OF InfCard_nat]]
unfolding Cofinite_def
      IsATopology_def by auto
    {
      assume AS:⋃MK={⋃T}
      moreover have ∀R∈MK. R⊆⋃MK by auto
      ultimately have ∀R∈MK. R⊆{⋃T} by auto
      then have ∀R∈MK. R={⋃T}∨R=0 by force moreover
      with sub have ∀R∈MK. R=0 by auto
      then have ⋃MK=0 by auto
      with AS have False by auto
    }
    with CO have CO2:⋃MK∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
by auto
    {
      assume ⋃MK∈(CoFinite (⋃T))
      then have ⋃MK∈T using assms(1) T1_cocardinal_coarser by auto
      with MN have {⋃MT,⋃MK}⊆(T) by auto
      then have (⋃MT)∪(⋃MK)∈T using union_open[OF topSpaceAssum, of
{⋃MT,⋃MK}] by auto
      then have ⋃M∈T using MM by auto
    }
    moreover
    {
```

```
      assume ⋃MK∉(CoFinite (⋃T))
      with CO obtain B where B{is compact in}(CoFinite (⋃T))B{is closed
in}(CoFinite (⋃T))
        ⋃MK={⋃CoFinite ⋃T}∪(⋃(CoFinite ⋃T)-B) unfolding OPCompactification_def
by auto
      then have MK:⋃MK={⋃T}∪(⋃T-B)B{is closed in}(CoFinite (⋃T))
        using union_cocardinal unfolding Cofinite_def by auto
      then have B:B⊆⋃T B≺nat∨B=⋃T using closed_sets_cocardinal un-
folding Cofinite_def by auto
      {
        assume B=⋃T
        with MK have ⋃MK={⋃T} by auto
        then have False using CO2 by auto
      }
      with B have B⊆⋃T and natB:B≺nat by auto
      have (⋃T-(⋃MT))∩B⊆B by auto
      then have (⋃T-(⋃MT))∩B≲B using subset_imp_lepoll by auto
      then have (⋃T-(⋃MT))∩B≺nat using natB lesspoll_trans1 by auto
      then have ((⋃T-(⋃MT))∩B){is closed in}(CoFinite (⋃T)) using
closed_sets_cocardinal
        B(1) unfolding Cofinite_def by auto
      then have ⋃T-((⋃T-(⋃MT))∩B)∈(CoFinite (⋃T)) unfolding IsClosed_def
using union_cocardinal unfolding Cofinite_def by auto
      also have ⋃T-((⋃T-(⋃MT))∩B)=(⋃T-(⋃T-(⋃MT)))∪(⋃T-B) by auto
      also have ...=(⋃MT)∪(⋃T-B) by auto
      ultimately have P:(⋃MT)∪(⋃T-B)∈(CoFinite (⋃T)) by auto
      then have eq:⋃T-(⋃T-((⋃MT)∪(⋃T-B)))=(⋃MT)∪(⋃T-B) by auto
      from P eq have (⋃T-((⋃MT)∪(⋃T-B))){is closed in}(CoFinite (⋃T))
unfolding IsClosed_def
        using union_cocardinal[of nat⋃T] unfolding Cofinite_def by auto
moreover
      have (⋃T-((⋃MT)∪(⋃T-B)))∩⋃T=(⋃T-((⋃MT)∪(⋃T-B))) by auto
      then have (CoFinite ⋃T){restricted to}(⋃T-((⋃MT)∪(⋃T-B)))=CoFinite
(⋃T-((⋃MT)∪(⋃T-B))) using subspace_cocardinal unfolding Cofinite_def
by auto
      then have (⋃T-((⋃MT)∪(⋃T-B))){is compact in}((CoFinite ⋃T){restricted
to}(⋃T-((⋃MT)∪(⋃T-B)))) using cofinite_compact
        union_cocardinal unfolding Cofinite_def by auto
      then have (⋃T-((⋃MT)∪(⋃T-B))){is compact in}(CoFinite ⋃T) us-
ing compact_subspace_imp_compact by auto ultimately
      have {⋃T}∪(⋃T-(⋃T-((⋃MT)∪(⋃T-B))))∈({one-point compactification
of}(CoFinite (⋃T)))
        unfolding OPCompactification_def using union_cocardinal unfold-
ing Cofinite_def by auto
      with eq have {⋃T}∪((⋃MT)∪(⋃T-B))∈({one-point compactification
of}(CoFinite (⋃T))) by auto
      moreover have AA:{⋃T}∪((⋃MT)∪(⋃T-B))=((⋃MT)∪(⋃MK)) using MK(1)
by auto
      ultimately have AA2:((⋃MT)∪(⋃MK))∈({one-point compactification
```

```
of}(CoFinite (⋃T))) by auto
      {
        assume AS:(⋃MT)∪(⋃MK)={⋃T}
        from MN have T:⋃T∉⋃MT using N by auto
        {
          fix x assume G:x∈⋃MT
          then have x∈(⋃MT)∪(⋃MK) by auto
          with AS have x∈{⋃T} by auto
          then have x=⋃T by auto
          with T have False using G by auto
        }
        then have ⋃MT=0 by auto
        with AS have (⋃MK)={⋃T} by auto
        then have False using CO2 by auto
      }
      with AA2 have ((⋃MT)∪(⋃MK))∈({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}} by auto
      with MM have ⋃M∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
by auto
    }
    ultimately
    have ⋃M∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
by auto
  }
  then have ∀M∈Pow((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T).
⋃M∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
    by auto moreover
  {
    fix U V assume U∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪TV∈(({one-
compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T moreover
    {
      assume U∈TV∈T
      then have U∩V∈T using topSpaceAssum unfolding IsATopology_def by
auto
      then have U∩V∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
by auto
    }
    moreover
    {
      assume UV:U∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})V∈(({one-poin
compactification of}(CoFinite (⋃T)))-{{⋃T}})
      then have O:U∩V∈({one-point compactification of}(CoFinite (⋃T)))
using topology0.op_comp_is_top[OF topology0_CoCardinal[OF InfCard_nat]]
unfolding Cofinite_def
        IsATopology_def by auto
      then have ⋃T∩(U∩V)∈({one-point compactification of}(CoFinite (⋃T))){restricted
to}⋃T
        unfolding RestrictedTo_def by auto
      then have ⋃T∩(U∩V)∈CoFinite ⋃T using topology0.open_subspace(2)[OF
```

```
topology0_CoCardinal[OF InfCard_nat]]
        union_cocardinal unfolding Cofinite_def by auto
      from UV have U≠{⋃T}V≠{⋃T}⋃T∩U∈({one-point compactification
of}(CoFinite (⋃T))){restricted to}⋃T⋃T∩V∈({one-point compactification
of}(CoFinite (⋃T))){restricted to}⋃T
        unfolding RestrictedTo_def by auto
      then have R:U≠{⋃T}V≠{⋃T}⋃T∩U∈CoFinite ⋃T⋃T∩V∈CoFinite ⋃T
using topology0.open_subspace(2)[OF topology0_CoCardinal[OF InfCard_nat]]
        union_cocardinal unfolding Cofinite_def by auto
      from UV have U⊆⋃({one-point compactification of}(CoFinite (⋃T)))V⊆⋃({one-point
compactification of}(CoFinite (⋃T))) by auto
      then have U⊆{⋃T}∪⋃TV⊆{⋃T}∪⋃T using topology0.op_compact_total[OF
topology0_CoCardinal[OF InfCard_nat]]
        union_cocardinal unfolding Cofinite_def by auto
      then have E:U=(⋃T∩U)∪({⋃T}∩U)V=(⋃T∩V)∪({⋃T}∩V)U∩V=(⋃T∩U∩V)∪({⋃T}∩U∩V)
by auto
      {
        assume Q:U∩V={⋃T}
        then have RR:⋃T∩(U∩V)=0 using N by auto
        {
          assume ⋃T∩U=0
          with E(1) have U={⋃T}∩U by auto
          also have ...⊆{⋃T} by auto
          ultimately have U⊆{⋃T} by auto
          then have U=0∨U={⋃T} by auto
          with R(1) have U=0 by auto
          then have U∩V=0 by auto
          then have False using Q by auto
        }
        moreover
        {
          assume ⋃T∩V=0
          with E(2) have V={⋃T}∩V by auto
          also have ...⊆{⋃T} by auto
          ultimately have V⊆{⋃T} by auto
          then have V=0∨V={⋃T} by auto
          with R(2) have V=0 by auto
          then have U∩V=0 by auto
          then have False using Q by auto
        }
        moreover
        {
          assume ⋃T∩U≠0⋃T∩V≠0
          with R(3,4) have (⋃T∩U)∩(⋃T∩V)≠0 using Cofinite_nat_HConn[OF
assms(2)]
            unfolding IsHConnected_def by auto
          then have ⋃T∩(U∩V)≠0 by auto
          then have False using RR by auto
        }
```

```
        ultimately have False by auto
      }
      with O have U∩V∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
by auto
    }
    moreover
    {
      assume UV:U∈TV∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
      from UV(2) obtain B where V∈(CoFinite ⋃T)∨(V={⋃T}∪(⋃T-B)∧B{is
closed in}(CoFinite (⋃T))) unfolding OPCompactification_def
        using union_cocardinal unfolding Cofinite_def by auto
      with assms(1) have V∈T∨(V={⋃T}∪(⋃T-B)∧B{is closed in}(CoFinite
(⋃T))) using T1_cocardinal_coarser by auto
      then have V∈T∨(U∩V=U∩(⋃T-B)∧B{is closed in}(CoFinite (⋃T)))
using UV(1) N by auto
      then have V∈T∨(U∩V=U∩(⋃T-B)∧(⋃T-B)∈(CoFinite (⋃T))) unfold-
ing IsClosed_def using union_cocardinal unfolding Cofinite_def by auto
      then have V∈T∨(U∩V=U∩(⋃T-B)∧(⋃T-B)∈T) using assms(1) T1_cocardinal_coarser
by auto
      with UV(1) have U∩V∈T using topSpaceAssum unfolding IsATopology_def
by auto
      then have U∩V∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
by auto
    }
    moreover
    {
      assume UV:U∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}V∈T
      from UV(1) obtain B where U∈(CoFinite ⋃T)∨(U={⋃T}∪(⋃T-B)∧B{is
closed in}(CoFinite (⋃T))) unfolding OPCompactification_def
        using union_cocardinal unfolding Cofinite_def by auto
      with assms(1) have U∈T∨(U={⋃T}∪(⋃T-B)∧B{is closed in}(CoFinite
(⋃T))) using T1_cocardinal_coarser by auto
      then have U∈T∨(U∩V=(⋃T-B)∩V∧B{is closed in}(CoFinite (⋃T)))
using UV(2) N by auto
      then have U∈T∨(U∩V=(⋃T-B)∩V∧(⋃T-B)∈(CoFinite (⋃T))) unfold-
ing IsClosed_def using union_cocardinal unfolding Cofinite_def by auto
      then have U∈T∨(U∩V=(⋃T-B)∩V∧(⋃T-B)∈T) using assms(1) T1_cocardinal_coarser
by auto
      with UV(2) have U∩V∈T using topSpaceAssum unfolding IsATopology_def
by auto
      then have U∩V∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
by auto
    }
    ultimately
    have U∩V∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
by auto
  }
  ultimately show thesis unfolding IsATopology_def by auto
qed
```

The previous construction preserves anti-hyperconnectedness.

```
theorem (in topology0) COF_comp_antiHConn:
  assumes T{is anti-}IsHConnected ¬(⋃T≺nat)
  shows ((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)
{is anti-}IsHConnected
proof-
  have N:⋃T∉(⋃T) using mem_not_refl by auto
  from assms(1) have T1:T{is T_1} using anti_HConn_imp_T1 by auto
  have tot1:⋃({one-point compactification of}(CoFinite (⋃T)))={⋃T}∪⋃T
using topology0.op_compact_total[OF topology0_CoCardinal[OF InfCard_nat],
of ⋃T]
        union_cocardinal[of nat⋃T] unfolding Cofinite_def by auto
  then have (⋃({one-point compactification of}(CoFinite (⋃T))))∪⋃T={⋃T}∪⋃T
by auto moreover
  have ⋃(({one-point compactification of}(CoFinite (⋃T)))∪T)=(⋃({one-point
compactification of}(CoFinite (⋃T))))∪⋃T
    by auto
  ultimately have tot2:⋃(({one-point compactification of}(CoFinite (⋃T)))∪T)={⋃T}∪⋃T
by auto
  have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T))) us-
ing union_open[OF topology0.op_comp_is_top[OF topology0_CoCardinal[OF
InfCard_nat]],of {one-point compactification of}(CoFinite (⋃T))]
    tot1 unfolding Cofinite_def by auto moreover
  {
    assume ⋃T=0
    with assms(2) have ¬(0≺nat) by auto
    then have False unfolding lesspoll_def using empty_lepollI eqpoll_0_is_0
      eqpoll_sym by auto
  }
  then have ⋃T≠0 by auto
  with N have Not:¬(⋃T⊆{⋃T}) by auto
  {
    assume {⋃T}∪⋃T={⋃T} moreover
    have ⋃T⊆{⋃T}∪⋃T by auto ultimately
    have ⋃T⊆{⋃T} by auto
    with Not have False by auto
  }
  then have {⋃T}∪⋃T≠{⋃T} by auto ultimately
  have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
by auto
  then have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T
by auto
  then have {⋃T}∪⋃T⊆⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)
by auto moreover
  have ({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T⊆({one-point
compactification of}(CoFinite (⋃T)))∪T by auto
  then have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)⊆⋃(({one-point
compactification of}(CoFinite (⋃T)))∪T) by auto
  with tot2 have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)⊆{⋃T}∪⋃T
```

```
by auto
  ultimately have TOT:⋃(((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)={⋃T
by auto
  {
    fix A assume AS:A⊆⋃T ((((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}A) {is hyperconnected}
    from AS(1,2) have e0:((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}A=(((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}⋃T){restricted to}A
      using subspace_of_subspace[of A⋃T((({one-point compactification
of}(CoFinite (⋃T)))-{{⋃T}})∪T)] TOT by auto
    have e1:(((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted
to}(⋃T))=((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}){restricted
to}⋃T)∪(T{restricted to}⋃T)
      unfolding RestrictedTo_def by auto
    {
      fix A assume A∈T{restricted to}⋃T
      then obtain B where B∈TA=B∩⋃T unfolding RestrictedTo_def by auto
      then have A=B by auto
      with ‹B∈T› have A∈T by auto
    }
    then have T{restricted to}⋃T⊆T by auto moreover
      {
      fix A assume A∈T
      then have ⋃T∩A=A by auto
      with ‹A∈T› have A∈T{restricted to}⋃T unfolding RestrictedTo_def
by auto
    }
    ultimately have T{restricted to}⋃T=T by auto moreover
    {
      fix A assume A∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}){restricted
to}⋃T
      then obtain B where B∈({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}}⋃T∩B=A unfolding RestrictedTo_def by auto
      then have B∈({one-point compactification of}(CoFinite (⋃T)))⋃T∩B=A
by auto
      then have A∈({one-point compactification of}(CoFinite (⋃T))){restricted
to}⋃T unfolding RestrictedTo_def by auto
      then have A∈(CoFinite (⋃T)) using topology0.open_subspace(2)[OF
topology0_CoCardinal[OF InfCard_nat]]
        union_cocardinal unfolding Cofinite_def by auto
      with T1 have A∈T using T1_cocardinal_coarser by auto
    }
    then have (({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}){restricted
to}⋃T⊆T by auto
    moreover note e1 ultimately
    have ((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}} ∪
T) {restricted to} (⋃T)) =T by auto
    with e0 have ((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted
```

```
to}A=T{restricted to}A by auto
    with assms(1) AS have A{is in the spectrum of}IsHConnected unfold-
ing antiProperty_def by auto
  }
  then have reg:∀A∈Pow(⋃T). (((((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}A) {is hyperconnected}) ⟶(A{is in
the spectrum of}IsHConnected) by auto
  have ⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto
  then have P:⋃T∈((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)
by auto
  {
    fix B assume sub:B∈Pow(⋃T ∪{⋃T}) and hyp:((((({one-point compactification
of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted to}B) {is hyperconnected})
    from P have  subop:⋃T∩B∈(((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}B) unfolding RestrictedTo_def by auto
    with hyp have hypSub:((((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}B){restricted to}(⋃T∩B)){is hyperconnected}
using topology0.open_subspace_hyperconn
      topology0.Top_1_L4 COF_comp_is_top[OF T1 assms(2)] unfolding topology0_def
by auto
    from sub TOT have B ⊆ ⋃(({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}} ∪ T) by auto
    then have (((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted
to}(⋃T∩B))=(((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted
to}B){restricted to}(⋃T∩B)
      using subspace_of_subspace[of ⋃T∩BB((({one-point compactification
of}(CoFinite (⋃T)))-{{⋃T}})∪T)] by auto
    with hypSub have ((({one-point compactification of}(CoFinite ⋃T))
- {{⋃T}} ∪ T) {restricted to} (⋃T ∩ B)){is hyperconnected} by auto
    with reg have (⋃T∩B){is in the spectrum of}IsHConnected by auto
    then have le:⋃T∩B≲1 using HConn_spectrum by auto
    {
      fix x assume x:x∈⋃T∩B
      with le have sing:⋃T∩B={x} using lepoll_1_is_sing by auto
      {
        fix y assume y:y∈B
        then have y∈⋃T ∪{⋃T} using sub by auto
        with y have y∈⋃T∩B∨y=⋃T by auto
        with sing have y=x∨y=⋃T by auto
      }
      then have B⊆{x,⋃T} by auto
      with x have disj:B={x}∨B={x,⋃T} by auto
      {
        assume ⋃T∈B
        with disj have B:B={x,⋃T} by auto
        from sing subop have singOp:{x}∈(((({one-point compactification
of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted to}B)
          by auto
        have {x}{is closed in}(CoFinite ⋃T) using topology0.T1_iff_singleton_closed[OF
```

```
topology0_CoCardinal[OF InfCard_nat]] cocardinal_is_T1[OF InfCard_nat]
          x union_cocardinal unfolding Cofinite_def by auto
        moreover
        have Finite({x}) by auto
        then have spec:{x}{is in the spectrum of} (λT. (⋃T) {is compact
in}T) using compact_spectrum by auto
        have ((CoFinite ⋃T){restricted to}{x}){is a topology}⋃((CoFinite
⋃T){restricted to}{x})={x}
          using topology0.Top_1_L4[OF topology0_CoCardinal[OF InfCard_nat]]
unfolding RestrictedTo_def Cofinite_def
          using x union_cocardinal by auto
        with spec have {x}{is compact in}((CoFinite ⋃T){restricted to}{x})
unfolding Spec_def
          by auto
        then have {x}{is compact in}(CoFinite ⋃T) using compact_subspace_imp_compact
          by auto moreover note x
        ultimately have {⋃T}∪(⋃T-{x})∈{one-point compactification of}(CoFinite
(⋃T)) unfolding OPCompactification_def
          using union_cocardinal unfolding Cofinite_def by auto more-
over
        {
          assume A:{⋃T}∪(⋃T-{x})={⋃T}
          {
            fix y assume P:y∈⋃T-{x}
            then have y∈{⋃T}∪(⋃T-{x}) by auto
            then have y=⋃T using A by auto
            with N P have False by auto
          }
          then have ⋃T-{x}=0 by auto
          with x have ⋃T={x} by auto
          then have ⋃T≈1 using singleton_eqpoll_1 by auto moreover
          have 1≺nat using n_lesspoll_nat by auto
          ultimately have ⋃T≺nat using eq_lesspoll_trans by auto
          then have False using assms(2) by auto
        }
        ultimately have {⋃T}∪(⋃T-{x})∈({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}} by auto
        then have  {⋃T}∪(⋃T-{x})∈(((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T)) by auto
        then have B∩({⋃T}∪(⋃T-{x}))∈(((({one-point compactification
of}(CoFinite (⋃T)))-{{⋃T}})∪T){restricted to}B) unfolding RestrictedTo_def
by auto
        moreover have B∩({⋃T}∪(⋃T-{x}))={⋃T} using B by auto
        ultimately have {⋃T}∈(((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T){restricted to}B) by auto
        with singOp hyp N x have False unfolding IsHConnected_def by
auto
      }
      with disj have B={x} by auto
```

```
      then have B≈1 using singleton_eqpoll_1 by auto
      then have B≲1 using eqpoll_imp_lepoll by auto
    }
    then have ⋃T∩B≠0⟶B≲1 by blast
    moreover
    {
      assume ⋃T∩B=0
      with sub have B⊆{⋃T} by auto
      then have B≲{⋃T} using subset_imp_lepoll by auto
      then have B≲1 using singleton_eqpoll_1 lepoll_eq_trans by auto
    }
    ultimately have B≲1 by auto
    then have B{is in the spectrum of}IsHConnected using HConn_spectrum
by auto
  }
  then show thesis unfolding antiProperty_def using TOT by auto
qed
```

The previous construction, applied to a densely ordered topology, gives the desired counterexample. What happends is that every neighbourhood of ⋃T is dense; because there are no finite open sets, and hence meets every non-empty open set. In conclusion, ⋃T cannot be separated from other points by disjoint open sets.

Every open set that contains ⋃T is dense, when considering the order topology in a densely ordered set with more than two points.

```
theorem neigh_infPoint_dense:
  fixes T X r
  defines T_def:T ≡ (OrdTopology X r)
  assumes IsLinOrder(X,r) X{is dense with respect to}r
    ∃x y. x≠y∧x∈X∧y∈X U∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T
⋃T∈U
    V∈(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T V≠0
  shows U∩V≠0
proof
  have N:⋃T∉(⋃T) using mem_not_refl by auto
  have tot1:⋃({one-point compactification of}(CoFinite (⋃T)))={⋃T}∪⋃T
using topology0.op_compact_total[OF topology0_CoCardinal[OF InfCard_nat],
of ⋃T]
        union_cocardinal[of nat⋃T] unfolding Cofinite_def by auto
  then have (⋃({one-point compactification of}(CoFinite (⋃T))))∪⋃T={⋃T}∪⋃T
by auto moreover
  have ⋃(({one-point compactification of}(CoFinite (⋃T)))∪T)=(⋃({one-point
compactification of}(CoFinite (⋃T))))∪⋃T
    by auto
  ultimately have tot2:⋃(({one-point compactification of}(CoFinite (⋃T)))∪T)={⋃T}∪⋃T
by auto
  have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T))) us-
ing union_open[OF topology0.op_comp_is_top[OF topology0_CoCardinal[OF
```

```
InfCard_nat]],of {one-point compactification of}(CoFinite (⋃T))]
    tot1 unfolding Cofinite_def by auto moreover
  {
    assume ⋃T=0
    then have X=0 unfolding T_def using union_ordtopology[OF assms(2)]
assms(4) by auto
    then have False using assms(4) by auto
  }
  then have ⋃T≠0 by auto
  with N have Not:¬(⋃T⊆{⋃T}) by auto
  {
    assume {⋃T}∪⋃T={⋃T} moreover
    have ⋃T⊆{⋃T}∪⋃T by auto ultimately
    have ⋃T⊆{⋃T} by auto
    with Not have False by auto
  }
  then have {⋃T}∪⋃T≠{⋃T} by auto ultimately
  have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
by auto
  then have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T
by auto
  then have {⋃T}∪⋃T⊆⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)
by auto moreover
  have ({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T⊆({one-point
compactification of}(CoFinite (⋃T)))∪T by auto
  then have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)⊆⋃(({one-point
compactification of}(CoFinite (⋃T)))∪T) by auto
  with tot2 have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)⊆{⋃T}∪⋃T
by auto
  ultimately have TOT:⋃((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)={⋃
by auto
  assume A:U∩V=0
  with assms(6) have NN:⋃T∉V by auto
  with assms(7) have V∈(CoFinite ⋃T)∪T unfolding OPCompactification_def
using union_cocardinal
    unfolding Cofinite_def by auto
  moreover have T{is T₂} unfolding T_def using order_top_T2[OF assms(2)]
assms(4) by auto
  then have T1:T{is T₁} using T2_is_T1 by auto
  ultimately have VopT:V∈T using topology0.T1_cocardinal_coarser[OF topology0_ordtopology(1
assms(2)]]
    unfolding T_def by auto
  from A assms(7) have V⊆⋃((({one-point compactification of}(CoFinite
(⋃T)))-{{⋃T}})∪T)-U by auto
  then have V⊆({⋃T}∪⋃T)-U using TOT by auto
  then have V⊆(⋃T)-U using NN by auto
  from N have U∉T using assms(6) by auto
  then have U∉(CoFinite ⋃T)∪T using T1 topology0.T1_cocardinal_coarser[OF
topology0_ordtopology(1)[OF assms(2)]]
```

```
    unfolding T_def using union_cocardinal union_ordtopology[OF assms(2)]
assms(4) by auto
  with assms(5,6) obtain B where U:U={⋃T}∪(⋃T-B) B{is closed in}(CoFinite
⋃T) B≠⋃T
    unfolding OPCompactification_def using union_cocardinal unfolding
Cofinite_def by auto
  then have U={⋃T}∪(⋃T-B) B=⋃T ∨ B≺nat B≠⋃T using closed_sets_cocardinal
unfolding Cofinite_def
    by auto
  then have U={⋃T}∪(⋃T-B) B≺nat by auto
  with N have ⋃T-U=⋃T-(⋃T-B) by auto
  then have ⋃T-U=B using U(2) unfolding IsClosed_def using union_cocardinal
unfolding Cofinite_def
    by auto
  with ‹B≺nat› have Finite(⋃T-U) using lesspoll_nat_is_Finite by auto
  with ‹V⊆(⋃T)-U› have Finite(V) using subset_Finite by auto
  from assms(8) obtain v where v∈V by auto
  with VopT have ∃R∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X} ∪ {LeftRayX(X,
r, b) . b ∈ X} ∪{RightRayX(X, r, b) . b ∈ X}. R ⊆ V ∧ v ∈ R using
    point_open_base_neigh[OF Ordtopology_is_a_topology(2)[OF assms(2)]]
unfolding T_def by auto
  then obtain R where R_def:R∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X}
∪ {LeftRayX(X, r, b) . b ∈ X} ∪{RightRayX(X, r, b) . b ∈ X} R⊆V v∈R
by blast
  moreover
  {
    assume R∈{IntervalX(X, r, b, c) . ⟨b,c⟩ ∈ X × X}
    then obtain b c where lim:b∈Xc∈XR=IntervalX(X, r, b, c) by auto
    with ‹v∈R› have  ¬ Finite(R) using dense_order_inf_intervals[OF assms(2)
_ _ _ assms(3)]
      by auto
    with ‹R⊆V› ‹Finite(V)› have False using subset_Finite by auto
  } moreover
  {
    assume R∈{LeftRayX(X, r, b) . b ∈ X}
    then obtain b where lim:b∈XR=LeftRayX(X, r, b) by auto
    with ‹v∈R› have  ¬ Finite(R) using dense_order_inf_lrays[OF assms(2)
_ _ assms(3)] by auto
    with ‹R⊆V› ‹Finite(V)› have False using subset_Finite by auto
  } moreover
  {
    assume R∈{RightRayX(X, r, b) . b ∈ X}
    then obtain b where lim:b∈XR=RightRayX(X, r, b) by auto
    with ‹v∈R› have  ¬ Finite(R) using dense_order_inf_rrays[OF assms(2)_
_ assms(3)] by auto
    with ‹R⊆V› ‹Finite(V)› have False using subset_Finite by auto
  } ultimately
  show False by auto
qed
```

A densely ordered set with more than one point gives an order topology. Applying the previous construction to this topology we get a non locally-Hausdorff space.

```
theorem OPComp_cofinite_dense_order_not_loc_T2:
  fixes T X r
  defines T_def:T ≡ (OrdTopology X r)
  assumes IsLinOrder(X,r) X{is dense with respect to}r
    ∃x y. x≠y∧x∈X∧y∈X
  shows ¬((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T){is
locally-T₂})
proof
  have N:⋃T∉(⋃T) using mem_not_refl by auto
  have tot1:⋃({one-point compactification of}(CoFinite (⋃T)))={⋃T}∪⋃T
using topology0.op_compact_total[OF topology0_CoCardinal[OF InfCard_nat],
of ⋃T]
        union_cocardinal[of nat⋃T] unfolding Cofinite_def by auto
  then have (⋃({one-point compactification of}(CoFinite (⋃T))))∪⋃T={⋃T}∪⋃T
by auto moreover
  have ⋃(({one-point compactification of}(CoFinite (⋃T)))∪T)=(⋃({one-point
compactification of}(CoFinite (⋃T))))∪⋃T
    by auto
  ultimately have tot2:⋃(({one-point compactification of}(CoFinite (⋃T)))∪T)={⋃T}∪⋃T
by auto
  have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T))) us-
ing union_open[OF topology0.op_comp_is_top[OF topology0_CoCardinal[OF
InfCard_nat]],of {one-point compactification of}(CoFinite (⋃T))]
    tot1 unfolding Cofinite_def by auto moreover
  {
    assume ⋃T=0
    then have X=0 unfolding T_def using union_ordtopology[OF assms(2)]
assms(4) by auto
    then have False using assms(4) by auto
  }
  then have ⋃T≠0 by auto
  with N have Not:¬(⋃T⊆{⋃T}) by auto
  {
    assume {⋃T}∪⋃T={⋃T} moreover
    have ⋃T⊆{⋃T}∪⋃T by auto ultimately
    have ⋃T⊆{⋃T} by auto
    with Not have False by auto
  }
  then have {⋃T}∪⋃T≠{⋃T} by auto ultimately
  have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}
by auto
  then have {⋃T}∪⋃T∈({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T
by auto
  then have {⋃T}∪⋃T⊆⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)
by auto moreover
  have ({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T⊆({one-point
```

```
compactification of}(CoFinite (⋃T)))∪T by auto
  then have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)⊆⋃(({one-point
compactification of}(CoFinite (⋃T)))∪T) by auto
  with tot2 have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)⊆{⋃T}∪⋃T
by auto
  ultimately have TOT:⋃((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}})∪T)={⋃
by auto
  have T1:T{is T_1} using order_top_T2[OF assms(2,4)] T2_is_T1 unfold-
ing T_def by auto moreover
  from assms(4) obtain b c where B:b∈Xc∈Xb≠c by auto
  {
    assume ⟨b,c⟩∉r
    with assms(2) have ⟨c,b⟩∈r unfolding IsLinOrder_def IsTotal_def us-
ing ‹b∈X›‹c∈X› by auto
    with assms(3) B obtain z where z∈X-{b,c}⟨c,z⟩∈r⟨z,b⟩∈r unfolding
IsDense_def by auto
    then have IntervalX(X,r,c,b)≠0 unfolding IntervalX_def using Order_ZF_2_L1
by auto
    then have ¬(Finite(IntervalX(X,r,c,b))) using dense_order_inf_intervals[OF
assms(2) _ ‹c∈X›‹b∈X› assms(3)]
      by auto moreover
    have IntervalX(X,r,c,b)⊆X unfolding IntervalX_def by auto
    ultimately have ¬(Finite(X)) using subset_Finite by auto
    then have ¬(X≺nat) using lesspoll_nat_is_Finite by auto
  }
  moreover
  {
    assume ⟨b,c⟩∈r
    with assms(3) B obtain z where z∈X-{b,c}⟨b,z⟩∈r⟨z,c⟩∈r unfolding
IsDense_def by auto
    then have IntervalX(X,r,b,c)≠0 unfolding IntervalX_def using Order_ZF_2_L1
by auto
    then have ¬(Finite(IntervalX(X,r,b,c))) using dense_order_inf_intervals[OF
assms(2) _ ‹b∈X›‹c∈X› assms(3)]
      by auto moreover
    have IntervalX(X,r,b,c)⊆X unfolding IntervalX_def by auto
    ultimately have ¬(Finite(X)) using subset_Finite by auto
    then have ¬(X≺nat) using lesspoll_nat_is_Finite by auto
  }
  ultimately have ¬(X≺nat) by auto
  with T1 have top:(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T){is
a topology} using topology0.COF_comp_is_top[OF topology0_ordtopology[OF
assms(2)]] unfolding T_def
    using union_ordtopology[OF assms(2,4)] by auto
  assume (({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T){is
locally-T_2} moreover
  have ⋃T∈⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)
using TOT by auto
  moreover have ⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)∈(({one-poi
```

compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)
using top **unfolding** IsATopology_def **by** auto
ultimately have ∃c∈Pow(⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T)). ⋃T ∈ Interior(c, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T) ∧
((({one-point compactification of}CoFinite ⋃T) - {{⋃T}} ∪ T) {restricted to} c) {is T_2} **unfolding** IsLocallyT2_def IsLocally_def[OF top] **by** auto
then obtain C **where** C:C⊆⋃(({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T) ⋃T ∈ Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T) **and** T2:((({one-point compactification of}CoFinite ⋃T) - {{⋃T}} ∪ T) {restricted to} C) {is T_2}
by auto
have sub:Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)⊆C **using** topology0.Top_2_L1
top **unfolding** topology0_def **by** auto
have (((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}C){restricted to}(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))=((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))
using subspace_of_subspace[OF sub C(1)] **by** auto **moreover**
have (⋃((({one-point compactification of}CoFinite ⋃T) - {{⋃T}} ∪ T) {restricted to} C))⊆C **unfolding** RestrictedTo_def **by** auto
with C(1) **have** (⋃((({one-point compactification of}CoFinite ⋃T) - {{⋃T}} ∪ T) {restricted to} C))=C **unfolding** RestrictedTo_def **by** auto
with sub **have** pp:Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)∈Pow(⋃((({one-point compactification of}CoFinite ⋃T) - {{⋃T}} ∪ T) {restricted to} C)) **by** auto
ultimately have T2_2:((((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))){is T_2}
using T2_here[OF T2 pp] **by** auto
have top2:((((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))){is a topology}
using topology0.Top_1_L4 top **unfolding** topology0_def **by** auto
from C(2) pp **have** p1:⋃T∈⋃((((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)))
unfolding RestrictedTo_def **by** auto
from top topology0.Top_2_L2 **have** intOP:(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))∈(({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T **unfolding** topology0_def **by** auto
{
fix x **assume** x≠⋃T x∈⋃((((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)))
with p1 **have** ∃U∈(((({one-point compactification of}(CoFinite ⋃T))

```
- {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))). ∃V∈(((({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point
compactification of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))).
      x∈U∧⋃T∈V∧U∩V=0 using T2_2 unfolding isT2_def by auto
    then obtain U V where UV:U∈(((({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)))
      V∈(((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}})
∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}}) ∪ T)))
      U≠0⋃T∈VU∩V=0 by auto
    from UV(1) obtain UC where U=(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))∩UCUC∈(((({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))
      unfolding RestrictedTo_def by auto
    with top intOP have Uop:U∈(({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}}) ∪ T unfolding IsATopology_def by auto
    from UV(2) obtain VC where V=(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))∩VCVC∈(((({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))
      unfolding RestrictedTo_def by auto
    with top intOP have V∈(({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}}) ∪ T unfolding IsATopology_def by auto
    with UV(3-5) Uop neigh_infPoint_dense[OF assms(2-4),of VU] union_ordtopology[OF
assms(2,4)]
      have False unfolding T_def by auto
  }
  then have ⋃(((({one-point compactification of}(CoFinite ⋃T)) - {{⋃T}})
∪ T){restricted to}(Interior(C, (({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}}) ∪ T)))⊆{⋃T}
    by auto
  with p1 have ⋃(((({one-point compactification of}(CoFinite ⋃T)) -
{{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)))={⋃T}
    by auto
  with top2 have {⋃T}∈(((({one-point compactification of}(CoFinite ⋃T))
- {{⋃T}}) ∪ T){restricted to}(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T)))
    unfolding IsATopology_def by auto
  then obtain W where UT:{⋃T}=(Interior(C, (({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T))∩WW∈(({one-point compactification
of}(CoFinite ⋃T)) - {{⋃T}}) ∪ T
    unfolding RestrictedTo_def by auto
  from this(2) have (Interior(C, (({one-point compactification of}(CoFinite
⋃T)) - {{⋃T}}) ∪ T))∩W∈(({one-point compactification of}(CoFinite ⋃T))
- {{⋃T}}) ∪ T using intOP
    top unfolding IsATopology_def by auto
  with UT(1) have {⋃T}∈(({one-point compactification of}(CoFinite ⋃T))
```

```
- {{⋃T}}) ∪ T by auto
  then have {⋃T}∈T by auto
  with N show False by auto
qed
```

This topology, from the previous result, gives a counter-example for anti-hyperconnected implies locally-T_2.

```
theorem antiHConn_not_imp_loc_T2:
  fixes T X r
  defines T_def:T ≡ (OrdTopology X r)
  assumes IsLinOrder(X,r) X{is dense with respect to}r
    ∃x y. x≠y∧x∈X∧y∈X
  shows ¬(((({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T){is
locally-T2})
  and (({one-point compactification of}(CoFinite (⋃T)))-{{⋃T}}∪T){is
anti-}IsHConnected
  using OPComp_cofinite_dense_order_not_loc_T2[OF assms(2-4)] dense_order_infinite[OF
assms(2-4)] union_ordtopology[OF assms(2,4)]
  topology0.COF_comp_antiHConn[OF topology0_ordtopology[OF assms(2)] topology0.T2_imp_anti_I
topology0_ordtopology[OF assms(2)] order_top_T2[OF assms(2,4)]]]
  unfolding T_def by auto
```

Let's prove that T_2 spaces are locally-T_2, but that there are locally-T_2 spaces which aren't T_2. In conclusion $T_2 \Rightarrow$ locally -$T_2 \Rightarrow$ anti-hyperconnected; all implications proper.

```
theorem(in topology0) T2_imp_loc_T2:
  assumes T{is T2}
  shows T{is locally-T2}
proof-
  {
    fix x assume x∈⋃T
    {
      fix b assume b:b∈Tx∈b
      then have (T{restricted to}b){is T2} using T2_here assms by auto
moreover
      from b have x∈int(b) using Top_2_L3 by auto
      ultimately have ∃c∈Pow(b). x∈int(c)∧(T{restricted to}c){is T2}
by auto
    }
    then have ∀b∈T. x∈b ⟶(∃c∈Pow(b). x∈int(c)∧(T{restricted to}c){is
T2}) by auto
  }
  then show thesis unfolding IsLocallyT2_def IsLocally_def[OF topSpaceAssum]
by auto
qed
```

If there is a closed singleton, then we can consider a topology that makes this point doble.

```
theorem(in topology0) doble_point_top:
```

```
  assumes {m}{is closed in}T
  shows (T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) {is a topology}
proof-
  {
    fix M assume M:M⊆T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}
    let MT={V∈M. V∈T}
    let Mm={V∈M. V∉T}
    have unm:⋃M=(⋃MT)∪(⋃Mm) by auto
    have tt:⋃MT∈T using topSpaceAssum unfolding IsATopology_def by auto
    {
      assume Mm=0
      then have ⋃Mm=0 by auto
      with unm have ⋃M=(⋃MT) by auto
      with tt have ⋃M∈T by auto
      then have ⋃M∈T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
    }
    moreover
    {
      assume AS:Mm≠0
      then obtain V where V:V∈MV∉T by auto
      with M have V∈{(U - {m}) ∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by blast
      then obtain U W where U:V=(U-{m})∪{⋃T}∪W U∈Tm∈U W∈T by auto
      let U={⟨V,W⟩∈T×T. m∈V∧ (V-{m})∪{⋃T}∪W∈Mm}
      let fU={fst(B). B∈U}
      let sU={snd(B). B∈U}
      have fU⊆TsU⊆T by auto
      then have P:⋃fU∈T⋃sU∈T using topSpaceAssum unfolding IsATopology_def
by auto moreover
      have ⟨U,W⟩∈U using U V by auto
      then have m∈⋃fU by auto
      ultimately have s:⟨⋃fU,⋃sU⟩∈{V∈T. m∈V}×T by auto
      moreover have r:∀S. ∀R. S∈{V∈T. m∈V}⟶ R∈T⟶(S-{m})∪{⋃T}∪R∈{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}
        by auto
      ultimately have (⋃fU-{m})∪{⋃T}∪⋃sU∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T} by auto
      {
        fix v assume v∈⋃Mm
        then obtain V where v:v∈VV∈Mm by auto
        then have V:V∈MV∉T by auto
        with M have V∈{U - {m} ∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by blast
        then obtain U W where U:V=(U-{m})∪{⋃T}∪W U∈Tm∈U W∈T by auto
        with v(1) have v∈(U-{m})∪{⋃T}∪W by auto
        then have v∈U-{m}∨v=⋃T∨v∈W by auto
        then have (v∈U∧v≠m)∨v=⋃T∨v∈W by auto
        moreover from U V have ⟨U,W⟩∈U by auto
        ultimately have v∈((⋃fU)-{m})∪{⋃T}∪(⋃sU) by auto
      }
      then have ⋃Mm⊆((⋃fU)-{m})∪{⋃T}∪(⋃sU) by blast moreover
```

```
      {
        fix v assume v:v∈((⋃fU)-{m})∪{⋃T}∪(⋃sU)
        {
          assume v=⋃T
          then have v∈(U-{m})∪{⋃T}∪W by auto
          with ‹⟨U,W⟩∈U› have v∈⋃Mm by auto
        }
        moreover
        {
          assume v≠⋃Tv∉⋃sU
          with v have v∈((⋃fU)-{m}) by auto
          then have (v∈⋃fU∧v≠m) by auto
          then obtain W where (v∈W∧W∈fU∧v≠m) by auto
          then have v∈(W-{m})∪{⋃T} W∈fU by auto
          then obtain B where fst(B)=W B∈U v∈(W-{m})∪{⋃T} by blast
          then have v∈⋃Mm by auto
        }
        ultimately have v∈⋃Mm by auto
      }
      then have ((⋃fU)-{m})∪{⋃T}∪(⋃sU)⊆⋃Mm by auto
      ultimately have ⋃Mm=((⋃fU)-{m})∪{⋃T}∪(⋃sU) by auto
      then have ⋃M=((⋃fU)-{m})∪{⋃T}∪((⋃sU)∪(⋃MT)) using unm by auto
      moreover from P tt have (⋃sU)∪(⋃MT)∈T using topSpaceAssum
        union_open[OF topSpaceAssum, of {⋃sU,⋃MT}] by auto
      with s have ⟨⋃fU,(⋃sU)∪(⋃MT)⟩∈{V∈T. m∈V}×T by auto
      then have ((⋃fU)-{m})∪{⋃T}∪((⋃sU)∪(⋃MT))∈{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T} using r
        by auto
      ultimately have ⋃M∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
      then have ⋃M∈T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
    }
    ultimately
    have ⋃M∈T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
  }
  then have ∀M∈Pow(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). ⋃M∈T∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T} by auto
  moreover
  {
    fix A B assume ass:A∈T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}B∈T
∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}
    {
      assume A:A∈T
      {
        assume B∈T
        with A have A∩B∈T using topSpaceAssum unfolding IsATopology_def
by auto
      }
      moreover
      {
```

```
        assume B∉T
        with ass(2) have B∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by
auto
        then obtain U W where U:U∈Tm∈UW∈TB=(U-{m})∪{⋃T}∪W by auto
moreover
        from A mem_not_refl have ⋃T∉A by auto
        ultimately have A∩B=A∩((U-{m})∪W) by auto
        then have eq:A∩B=(A∩(U-{m}))∪(A∩W) by auto
        have ⋃T-{m}∈T using assms unfolding IsClosed_def by auto
        with U(1) have O:U∩(⋃T-{m})∈T using topSpaceAssum unfolding
IsATopology_def
          by auto
        have U∩(⋃T-{m})=U-{m} using U(1) by auto
        with O have U-{m}∈T by auto
        with A have (A∩(U-{m}))∈T using topSpaceAssum unfolding IsATopology_def
          by auto
        moreover
        from A U(3) have A∩W∈T using topSpaceAssum unfolding IsATopology_def
          by auto
        ultimately have (A∩(U-{m}))∪(A∩W)∈T using
          union_open[OF topSpaceAssum, of {A∩(U-{m}),A∩W}] by auto
        with eq have A∩B∈T by auto
      }
      ultimately have A∩B∈T by auto
    }
    moreover
    {
      assume A∉T
      with ass(1) have A:A∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by
auto
        {
        assume B:B∈T
        from A obtain U W where U:U∈Tm∈UW∈TA=(U-{m})∪{⋃T}∪W by auto
moreover
        from B mem_not_refl have ⋃T∉B by auto
        ultimately have A∩B=((U-{m})∪W)∩B by auto
        then have eq:A∩B=((U-{m})∩B)∪(W∩B) by auto
        have ⋃T-{m}∈T using assms unfolding IsClosed_def by auto
        with U(1) have O:U∩(⋃T-{m})∈T using topSpaceAssum unfolding
IsATopology_def
          by auto
        have U∩(⋃T-{m})=U-{m} using U(1) by auto
        with O have U-{m}∈T by auto
        with B have ((U-{m})∩B)∈T using topSpaceAssum unfolding IsATopology_def
          by auto
        moreover
        from B U(3) have W∩B∈T using topSpaceAssum unfolding IsATopology_def
          by auto
        ultimately have ((U-{m})∩B)∪(W∩B)∈T using
```

```
      union_open[OF topSpaceAssum, of {((U-{m})∩B),(W∩B)}] by auto
    with eq have A∩B∈T by auto
  }
  moreover
  {
    assume B∉T
    with ass(2) have B∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by
auto
    then obtain U W where U:U∈Tm∈UW∈TB=(U-{m})∪{⋃T}∪W by auto
moreover
    from A obtain UA WA where UA:UA∈Tm∈UAWA∈TA=(UA-{m})∪{⋃T}∪WA
by auto
    ultimately have A∩B=(((UA-{m})∪WA)∩((U-{m})∪W))∪{⋃T} by auto
    then have eq:A∩B=((UA-{m})∩(U-{m}))∪(WA∩(U-{m}))∪((UA-{m})∩W)∪(WA∩W)∪{⋃T}
by auto
    have ⋃T-{m}∈T using assms unfolding IsClosed_def by auto
    with U(1) UA(1) have O:U∩(⋃T-{m})∈TUA∩(⋃T-{m})∈T using topSpaceAssum
unfolding IsATopology_def
      by auto
    have U∩(⋃T-{m})=U-{m}UA∩(⋃T-{m})=UA-{m} using U(1) UA(1) by
auto
    with O have OO:U-{m}∈TUA-{m}∈T by auto
    then have ((UA-{m})∩(U-{m}))=UA∩U-{m} by auto
    moreover
    have UA∩U∈Tm∈UA∩U using U(1,2) UA(1,2) topSpaceAssum unfold-
ing IsATopology_def
      by auto
    moreover
    from OO U(3) UA(3) have TT:WA∩(U-{m})∈T(UA-{m})∩W∈TWA∩W∈T us-
ing topSpaceAssum unfolding IsATopology_def
      by auto
    from TT(2,3) have ((UA-{m})∩W)∪(WA∩W)∈T using union_open[OF
topSpaceAssum,
      of {(UA-{m})∩W,WA∩W}] by auto
    with TT(1) have (WA∩(U-{m}))∪(((UA-{m})∩W)∪(WA∩W))∈T using union_open[OF
topSpaceAssum,
      of {WA∩(U-{m}),((UA-{m})∩W)∪(WA∩W)}] by auto
    ultimately
    have A∩B=(UA∩U-{m})∪{⋃T}∪((WA∩(U-{m}))∪(((UA-{m})∩W)∪(WA∩W)))
      (WA∩(U-{m}))∪(((UA-{m})∩W)∪(WA∩W))∈T UA∩U∈{V∈T. m∈V} using
eq by auto
    then have ∃W∈T. A∩B=(UA∩U-{m})∪{⋃T}∪W UA∩U∈{V∈T. m∈V} by
auto
    then have A∩B∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
  }
  ultimately
  have A∩B∈T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
}
ultimately have A∩B∈T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
```

```
  }
  then have ∀A∈T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}. ∀B∈T∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}.
    A∩B∈T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by blast
  ultimately show thesis unfolding IsATopology_def by auto
qed
```

The previous topology is defined over a set with one more point.

```
lemma(in topology0) union_doublepoint_top:
  assumes {m}{is closed in}T
  shows ⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})=⋃T ∪{⋃T}
proof
  {
    fix x assume x∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
    then obtain R where x:x∈RR∈T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}
by blast
    {
      assume R∈T
      with x(1) have x∈⋃T by auto
    }
    moreover
    {
      assume R∉T
      with x(2) have R∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
      then obtain U W where R=(U-{m})∪{⋃T}∪WW∈TU∈Tm∈U by auto
      with x(1) have x=⋃T∨x∈⋃T by auto
    }
    ultimately have x∈⋃T ∪{⋃T} by auto
  }
  then show ⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})⊆⋃T ∪{⋃T}
by auto
  {
    fix x assume x∈⋃T ∪{⋃T}
    then have dis:x∈⋃T∨x=⋃T by auto
    {
      assume x∈⋃T
      then have x∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) by auto
    }
    moreover
    {
      assume x∉⋃T
      with dis have x=⋃T by auto
      moreover from assms have ⋃T-{m}∈Tm∈⋃T unfolding IsClosed_def
by auto
      moreover have 0∈T using empty_open topSpaceAssum by auto
      ultimately have x∈(⋃T-{m})∪{⋃T}∪0 (⋃T-{m})∪{⋃T}∪0∈{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}
        using union_open[OF topSpaceAssum] by auto
      then have x∈(⋃T-{m})∪{⋃T}∪0 (⋃T-{m})∪{⋃T}∪0∈T ∪{(U-{m})∪{⋃T}∪W.
```

```
⟨U,W⟩∈{V∈T. m∈V}×T}
        by auto
      then have x∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) by blast
    }
    ultimately have x∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) by
auto
  }
  then show ⋃T ∪{⋃T}⊆⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
by auto
qed
```

In this topology, the previous topological space is an open subspace.

```
theorem(in topology0) open_subspace_double_point:
  assumes {m}{is closed in}T
  shows (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}⋃T=T
and ⋃T∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
proof-
  have N:⋃T∉⋃T using mem_not_refl by auto
  {
    fix x assume x∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}⋃T
    then obtain U where U:U∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})x=⋃T∩U
      unfolding RestrictedTo_def by blast
    {
      assume U∉T
      with U(1) have  U∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
      then obtain V W where VW:U=(V-{m})∪{⋃T}∪WV∈Tm∈VW∈T by auto
      with N U(2) have x:x=(V-{m})∪W by auto
      have ⋃T-{m}∈T using assms unfolding IsClosed_def by auto
      then have V∩(⋃T-{m})∈T using VW(2) topSpaceAssum unfolding IsATopology_def
        by auto moreover
      have V-{m}=V∩(⋃T-{m}) using VW(2,3) by auto ultimately
      have V-{m}∈T by auto
      with VW(4) have (V-{m})∪W∈T using union_open[OF topSpaceAssum,
of {V-{m},W}]
        by auto
      with x have x∈T by auto
    }
    moreover
    {
      assume A:U∈T
      with U(2) have x=U by auto
      with A have x∈T by auto
    }
    ultimately have x∈T by auto
  }
  then have (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}⋃T⊆T
by auto
  moreover
```

```
  {
    fix x assume x:x∈T
    then have x∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) by auto more-
over
    from x have ⋃T∩x=x by auto ultimately
    have ∃M∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). ⋃T∩M=x by blast
    then have x∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}⋃T unfolding RestrictedTo_def
      by auto
  }
  ultimately show (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}⋃T=T by auto
  have P:⋃T∈T using topSpaceAssum unfolding IsATopology_def by auto
  then show ⋃T∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) by auto
qed
```

The previous topology construction applied to a T_2 non-discrite space topology, gives a counter-example to: Every locally-T_2 space is T_2.

If there is a singleton which is not open, but closed; then the construction on that point is not T_2.

```
theorem(in topology0) loc_T2_imp_T2_counter_1:
  assumes {m}∉T {m}{is closed in}T
  shows ¬((T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) {is T2})
proof
  assume ass:(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) {is T2}
  then have tot1:⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})=⋃T ∪{⋃T}
using union_doublepoint_top
    assms(2) by auto
  have m≠⋃T using mem_not_refl assms(2) unfolding IsClosed_def by auto
moreover
  from ass tot1 have ∀x y. x∈⋃T ∪{⋃T} ∧ y∈⋃T ∪{⋃T}∧x≠y ⟶ (∃𝔘∈(T∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}).
    ∃𝔙∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). x∈𝔘∧y∈𝔙∧𝔘∩𝔙=0)
unfolding isT2_def by auto
  moreover
  from assms(2) have m∈⋃T ∪{⋃T} unfolding IsClosed_def by auto more-
over
  have ⋃T∈⋃T ∪{⋃T} by auto ultimately
  have ∃𝔘∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). ∃𝔙∈(T∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}). m∈𝔘∧⋃T∈𝔙∧𝔘∩𝔙=0
    by auto
  then obtain 𝔘 𝔙 where UV:𝔘∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
    𝔙∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})m∈𝔘⋃T∈𝔙𝔘∩𝔙=0 using
tot1 by blast
  then have ⋃T∉𝔘 by auto
  with UV(1) have P:𝔘∈T by auto
  {
    assume 𝔙∈T
```

```
    then have 𝔙⊆⋃T by auto
    with UV(4) have ⋃T∈⋃T using tot1 by auto
    then have False using mem_not_refl by auto
  }
  with UV(2) have 𝔙∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by auto
  then obtain U W where V:𝔙=(U-{m})∪{⋃T}∪W U∈Tm∈UW∈T by auto
  from V(2,3) P have int:U∩𝔘∈Tm∈U∩𝔘 using UV(3) topSpaceAssum
    unfolding IsATopology_def by auto
  have (U∩𝔘-{m})⊆𝔘 (U∩𝔘-{m})⊆𝔙 using V(1) by auto
  then have (U∩𝔘-{m})=0 using UV(5) by auto
  with int(2) have U∩𝔘={m} by auto
  with int(1) assms(1) show False by auto
qed
```

This topology is locally-T_2.

```
theorem(in topology0) loc_T2_imp_T2_counter_2:
  assumes {m}∉T m∈⋃T T{is T₂}
  shows (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) {is locally-T₂}
proof-
  from assms(3) have T{is T₁} using T2_is_T1 by auto
  with assms(2) have mc:{m}{is closed in}T using T1_iff_singleton_closed
by auto
  have N:⋃T∉⋃T using mem_not_refl by auto
  have res:(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}⋃T=T
    and P:⋃T∈T and Q:⋃T∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
using open_subspace_double_point mc
    topSpaceAssum unfolding IsATopology_def by auto
  {
    fix A assume ass:A∈⋃T ∪{⋃T}
    {
      assume A≠⋃T
      with ass have A∈⋃T by auto
      with Q res assms(3) have ⋃T∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})∧
A∈⋃T ∧ (((T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}⋃T){is
T₂}) by auto
      then have ∃Z∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). A∈Z∧(((T∪{(U-{m})∪{⋃T}∪W
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}Z){is T₂})
        by blast
    }
    moreover
    {
      assume A:A=⋃T
      have ⋃T∈Tm∈⋃T0∈T using assms(2) empty_open[OF topSpaceAssum]
unfolding IsClosed_def using P by auto
      then have (⋃T-{m})∪{⋃T}∪0∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}
by auto
      then have opp:(⋃T-{m})∪{⋃T}∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}) by auto
      {
```

```
        fix A1 A2 assume points:A1∈(⋃T-{m})∪{⋃T}A2∈(⋃T-{m})∪{⋃T}A1≠A2
        from points(1,2) have notm:A1≠mA2≠m using assms(2) unfolding
IsClosed_def
          using mem_not_refl by auto
        {
          assume or:A1∈⋃TA2∈⋃T
          with points(3) assms(3) obtain U V where UV:U∈TV∈TA1∈UA2∈V
            U∩V=0 unfolding isT2_def by blast
          from UV(1,2) have U∩((⋃T-{m})∪{⋃T})∈(T∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})
            V∩((⋃T-{m})∪{⋃T})∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})
            unfolding RestrictedTo_def by auto moreover
          then have U∩(⋃T-{m})=U∩((⋃T-{m})∪{⋃T}) V∩(⋃T-{m})=V∩((⋃T-{m})∪{⋃T})
using UV(1,2) mem_not_refl[of ⋃T]
            by auto
          ultimately have opUV:U∩(⋃T-{m})∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})
            V∩(⋃T-{m})∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}) by auto
          moreover have U∩(⋃T-{m})∩(V∩(⋃T-{m}))=0 using UV(5) by auto
moreover
          from UV(3) or(1) notm(1) have A1∈U∩(⋃T-{m}) by auto more-
over
          from UV(4) or(2) notm(2) have A2∈V∩(⋃T-{m}) by auto ulti-
mately
          have ∃V. V∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})∧ A1∈U∩(⋃T-{m})∧A2∈V∧(U∩(⋃T-{m}))∩V=0 using exI[where
x=V∩(⋃T-{m}) and P=λW. W∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})∧A1∈(U∩(⋃T-{m}))∧A2∈W∧(U∩(⋃T-{m}))∩W=0]
            using opUV(2) by auto
          then have ∃U. U∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})∧(∃V. V∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})∧
            A1∈U∧A2∈V∧U∩V=0) using exI[where x=U∩(⋃T-{m}) and P=λW.
W∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})∧(∃V.
V∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})∧
A1∈W∧A2∈V∧W∩V=0)]
            using opUV(1) by auto
          then have ∃U∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}). (∃V. V∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})∧A1∈U∧A2∈V∧U∩V=0) by blast
          then have ∃U∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}). (∃V∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}). A1∈U∧A2∈V∧U∩V=0) by blast
        }
        moreover
        {
          assume A1∉⋃T
```

```
          then have ig:A1=⋃T using points(1) by auto
          {
            assume A2∉⋃T
            then have A2=⋃T using points(2) by auto
            with points(3) ig have False by auto
          }
          then have igA2:A2∈⋃T by auto moreover
          have m∈⋃T using assms(2) unfolding IsClosed_def by auto
          moreover note notm(2) assms(3) ultimately obtain U V where
UV:U∈TV∈T
            m∈UA2∈VU∩V=0 unfolding isT2_def by blast
          from UV(1,3) have U∈{W∈T. m∈W} by auto moreover
          have 0∈T using empty_open topSpaceAssum by auto ultimately
          have (U-{m})∪{⋃T}∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by
auto
          then have Uop:(U-{m})∪{⋃T}∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}) by auto
          from UV(2) have Vop:V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
by auto
          from UV(1-3,5) have sub:V⊆(⋃T-{m})∪{⋃T} ((U-{m})∪{⋃T})⊆(⋃T-{m})∪{⋃T}
by auto
          from sub(1) have V=((⋃T-{m})∪{⋃T})∩V by auto
          then have VV:V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}) unfolding RestrictedTo_def
            using Vop by blast moreover
          from sub(2) have ((U-{m})∪{⋃T})=((⋃T-{m})∪{⋃T})∩((U-{m})∪{⋃T})
by auto
          then have UU:((U-{m})∪{⋃T})∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}) unfolding RestrictedTo_def
            using Uop by blast moreover
          from UV(2) have ((U-{m})∪{⋃T})∩V=(U-{m})∩V using mem_not_refl
by auto
          then  have ((U-{m})∪{⋃T})∩V=0 using UV(5) by auto
          with UV(4) VV ig igA2 have ∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
            A1∈(U-{m})∪{⋃T}∧A2∈V∧((U-{m})∪{⋃T})∩V=0 by auto
          with UU ig have ∃U. U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})∧ (∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}).
            A1∈U∧A2∈V∧U∩V=0) using exI[where x=((U-{m})∪{⋃T}) and
P=λU. U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})∧
(∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
            A1∈U∧A2∈V∧U∩V=0)] by auto
          then have ∃U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}). (∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}).
            A1∈U∧A2∈V∧U∩V=0) by blast
        }
        moreover
```

```
{
  assume A2∉⋃T
  then have ig:A2=⋃T using points(2) by auto
  {
    assume A1∉⋃T
    then have A1=⋃T using points(1) by auto
    with points(3) ig have False by auto
  }
  then have igA2:A1∈⋃T by auto moreover
  have m∈⋃T using assms(2) unfolding IsClosed_def by auto
  moreover note notm(1) assms(3) ultimately obtain U V where
UV:U∈TV∈T
    m∈UA1∈VU∩V=0 unfolding isT2_def by blast
  from UV(1,3) have U∈{W∈T. m∈W} by auto moreover
  have 0∈T using empty_open topSpaceAssum by auto ultimately
  have (U-{m})∪{⋃T}∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} by
auto
  then have Uop:(U-{m})∪{⋃T}∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}) by auto
  from UV(2) have Vop:V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
by auto
  from UV(1-3,5) have sub:V⊆(⋃T-{m})∪{⋃T} ((U-{m})∪{⋃T})⊆(⋃T-{m})∪{⋃T}
by auto
  from sub(1) have V=((⋃T-{m})∪{⋃T})∩V by auto
  then have VV:V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}) unfolding RestrictedTo_def
    using Vop by blast moreover
  from sub(2) have ((U-{m})∪{⋃T})=((⋃T-{m})∪{⋃T})∩((U-{m})∪{⋃T})
by auto
  then have UU:((U-{m})∪{⋃T})∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}) unfolding RestrictedTo_def
    using Uop by blast moreover
  from UV(2) have V∩((U-{m})∪{⋃T})=V∩(U-{m}) using mem_not_refl
by auto
  then have V∩((U-{m})∪{⋃T})=0 using UV(5) by auto
  with UU UV(4) ig igA2 have ∃U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
    A1∈V∧A2∈U∧V∩U=0 by auto
  with VV igA2 have ∃U. U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})∧ (∃V∈(T ∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
    A1∈U∧A2∈V∧U∩V=0) using exI[where x=V and P=λU. U∈(T ∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T})∧ (∃V∈(T ∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
    A1∈U∧A2∈V∧U∩V=0)] by auto
  then have ∃U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}). (∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}).
    A1∈U∧A2∈V∧U∩V=0) by blast
```

```
        }
        ultimately have ∃U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}). (∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}).
            A1∈U∧A2∈V∧U∩V=0) by blast
      }
      then have ∀A1∈(⋃T-{m})∪{⋃T}. ∀A2∈(⋃T-{m})∪{⋃T}. A1≠A2 ⟶
(∃U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
(∃V∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
            A1∈U∧A2∈V∧U∩V=0)) by auto moreover
      have ⋃((T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}))=(⋃(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))∩((⋃T-{m})∪{⋃T})
        unfolding RestrictedTo_def by auto
      then have ⋃((T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}))=(⋃T ∪{⋃T})∩((⋃T-{m})∪{⋃T}) using
        union_doublepoint_top mc by auto
      then have ⋃((T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T}))=(⋃T-{m})∪{⋃T} by auto
      ultimately have ∀A1∈⋃((T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})). ∀A2∈⋃((T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})). A1≠A2 ⟶ (∃U∈(T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}). (∃V∈(T ∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}((⋃T-{m})∪{⋃T}).
            A1∈U∧A2∈V∧U∩V=0)) by auto
      then have ((T ∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}((⋃T-{m})∪{⋃T})){is T₂} unfolding isT2_def
        by force
      with opp A have ∃Z∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}).
A∈Z∧(((T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}Z){is
T₂})
        by blast
    }
    ultimately
    have ∃Z∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). A∈Z∧(((T∪{(U-{m})∪{⋃T}∪W.
⟨U,W⟩∈{V∈T. m∈V}×T}){restricted to}Z){is T₂})
        by blast
  }
  then have ∀A∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). ∃Z∈T ∪
{U - {m} ∪ {⋃T} ∪ W . ⟨U,W⟩ ∈ {V ∈ T . m ∈ V} × T}.
     A ∈ Z ∧ ((T ∪ {U - {m} ∪ {⋃T} ∪ W . ⟨U,W⟩ ∈ {V ∈ T . m ∈ V} ×
T}) {restricted to} Z) {is T₂}
     using union_doublepoint_top mc by auto
  with topology0.loc_T2 show (T ∪ {U - {m} ∪ {⋃T} ∪ W . ⟨U,W⟩ ∈ {V ∈
T . m ∈ V} × T}){is locally-T₂}
    unfolding topology0_def using doble_point_top mc by auto
qed
```

There can be considered many more local properties, which; as happens with locally-T_2; can distinguish between spaces other properties cannot.

end

89 Properties in Topology 3

```
theory Topology_ZF_properties_3 imports Topology_ZF_7 Finite_ZF_1 Topology_ZF_1b
Topology_ZF_9
  Topology_ZF_properties_2 FinOrd_ZF
begin
```

This theory file deals with more topological properties and the relation with the previous ones in other theory files.

89.1 More anti-properties

In this section we study more anti-properties.

89.2 First examples

A first example of an anti-compact space is the discrete space.

```
lemma pow_compact_imp_finite:
  assumes B{is compact in}Pow(A)
  shows Finite(B)
proof-
  from assms have B:B⊆A ∀M∈Pow(Pow(A)). B⊆⋃M ⟶(∃N∈FinPow(M). B⊆⋃N)
    unfolding IsCompact_def by auto
  from B(1) have {{x}. x∈B}∈Pow(Pow(A)) B⊆⋃{{x}. x∈B} by auto
  with B(2) have ∃N∈FinPow({{x}. x∈B}). B⊆⋃N by auto
  then obtain N where N∈FinPow({{x}. x∈B}) B⊆⋃N by auto
  then have Finite(N) N⊆{{x}. x∈B} B⊆⋃N unfolding FinPow_def by auto
  then have Finite(N) ∀b∈N. Finite(b) B⊆⋃N by auto
  then have B⊆⋃N Finite(⋃N) using Finite_Union[of N] by auto
  then show Finite(B) using subset_Finite by auto
qed

theorem pow_anti_compact:
  shows Pow(A){is anti-compact}
proof-
  {
    fix B assume as:B⊆⋃Pow(A) (⋃(Pow(A){restricted to}B)){is compact
in}(Pow(A){restricted to}B)
    then have sub:B⊆A by auto
    then have Pow(B)=Pow(A){restricted to}B unfolding RestrictedTo_def
by blast
    with as(2) have (⋃Pow(B)){is compact in}Pow(B) by auto
    then have B{is compact in}Pow(B) by auto
    then have Finite(B) using pow_compact_imp_finite by auto
```

```
    then have B{is in the spectrum of}(λT. (⋃T){is compact in}T) us-
ing compact_spectrum by auto
  }
  then show thesis unfolding IsAntiComp_def antiProperty_def by auto
qed
```

In a previous file, `Topology_ZF_5.thy`, we proved that the spectrum of the lindelöf property depends on the axiom of countable choice on subsets of the power set of the natural number.

In this context, the examples depend on wether this choice principle holds or not. This is the reason that the examples of anti-lindeloef topologies are left for the next section.

89.3 Structural results

We first differenciate the spectrum of the lindeloef property depending on some axiom of choice.

```
lemma lindeloef_spec1:
  assumes {the axiom of} nat {choice holds for subsets}(Pow(nat))
  shows (A {is in the spectrum of} (λT. ((⋃T){is lindeloef in}T))) ⟷
(A≲nat)
  using compactK_spectrum[OF assms Card_nat] unfolding IsLindeloef_def.

lemma lindeloef_spec2:
  assumes ¬({the axiom of} nat {choice holds for subsets}(Pow(nat)))
  shows (A {is in the spectrum of} (λT. ((⋃T){is lindeloef in}T))) ⟷
Finite(A)
proof
  assume Finite(A)
  then have A:A{is in the spectrum of} (λT. ((⋃T){is compact in}T))
using compact_spectrum by auto
  have s:nat≲csucc(nat) using le_imp_lesspoll[OF Card_csucc[OF Ord_nat]]
lt_csucc[OF Ord_nat] le_iff by auto
  {
    fix T assume T{is a topology} (⋃T){is compact in}T
    then have (⋃T){is compact of cardinal}nat{in}T using Compact_is_card_nat
by auto
    then have (⋃T){is compact of cardinal}csucc(nat){in}T using s compact_greater_card
Card_csucc[OF Ord_nat] by auto
    then have (⋃T){is lindeloef in}T unfolding IsLindeloef_def by auto
  }
  then have ∀T. T{is a topology} ⟶ ((⋃T){is compact in}T) ⟶ ((⋃T){is
lindeloef in}T) by auto
  with A show A {is in the spectrum of} (λT. ((⋃T){is lindeloef in}T))
using P_imp_Q_spec_inv[
    where Q=λT. ((⋃T){is compact in}T) and P=λT. ((⋃T){is lindeloef
in}T)] by auto
```

```
next
  assume A:A {is in the spectrum of} (λT. ((⋃T){is lindeloef in}T))
  then have reg:∀T. T{is a topology}∧⋃T≈A ⟶ ((⋃T){is compact of
cardinal} csucc(nat){in}T) using Spec_def
    unfolding IsLindeloef_def by auto
  then have A{is compact of cardinal} csucc(nat) {in} Pow(A) using Pow_is_top[of
A] by auto
  then have ∀M∈Pow(Pow(A)). A⊆⋃M ⟶ (∃N∈Pow(M). A⊆⋃N ∧ N≺csucc(nat))
unfolding IsCompactOfCard_def by auto
  moreover
  have {{x}. x∈A}∈Pow(Pow(A)) by auto
  moreover
  have A=⋃{{x}. x∈A} by auto
  ultimately have ∃N∈Pow({{x}. x∈A}). A⊆⋃N ∧ N≺csucc(nat) by auto
  then obtain N where N∈Pow({{x}. x∈A}) A⊆⋃N N≺csucc(nat) by auto
  then have N⊆{{x}. x∈A} N≺csucc(nat) A⊆⋃N using FinPow_def by auto
  {
    fix t
    assume t∈{{x}. x∈A}
    then obtain x where x∈At={x} by auto
    with ‹A⊆⋃N› have x∈⋃N by auto
    then obtain B where B∈Nx∈B by auto
    with ‹N⊆{{x}. x∈A}› have B={x} by auto
    with ‹t={x}›‹B∈N› have t∈N by auto
  }
  with ‹N⊆{{x}. x∈A}› have N={{x}. x∈A} by auto
  let B={⟨x,{x}⟩. x∈A}
  from ‹N={{x}. x∈A}› have B:A→ N unfolding Pi_def function_def by auto
  with ‹N={{x}. x∈A}› have B:inj(A,N) unfolding inj_def using apply_equality
by auto
  then have A≲N using lepoll_def by auto
  with ‹N≺csucc(nat)› have A≺csucc(nat) using lesspoll_trans1 by auto
  then have A≲nat using Card_less_csucc_eq_le Card_nat by auto
  then have A≺nat∨A≈nat using lepoll_iff_leqpoll by auto moreover
  {
    assume A≈nat
    then have nat≈A using eqpoll_sym by auto
    with A have nat {is in the spectrum of} (λT. ((⋃T){is lindeloef
in}T)) using equipollent_spect[
      where P=(λT. ((⋃T){is lindeloef in}T))] by auto
    moreover
    have Pow(nat){is a topology} using Pow_is_top by auto
    moreover
    have ⋃Pow(nat)=nat by auto
    then have ⋃Pow(nat)≈nat using eqpoll_refl by auto
    ultimately
    have nat {is compact of cardinal} csucc(nat){in}Pow(nat) using Spec_def
unfolding IsLindeloef_def by auto
    then have False using Q_disc_comp_csuccQ_eq_Q_choice_csuccQ[OF InfCard_nat]
```

```
assms by auto
  }
  ultimately have A≺nat by auto
  then show Finite(A) using lesspoll_nat_is_Finite by auto
qed
```

If the axiom of countable choice on subsets of the pow of the natural numbers doesn't hold, then anti-lindeloef spaces are anti-compact.

```
theorem(in topology0) no_choice_imp_anti_lindeloef_is_anti_comp:
  assumes ¬({the axiom of} nat {choice holds for subsets}(Pow(nat)) )
T{is anti-lindeloef}
  shows T{is anti-compact}
proof-
  have s:nat≲csucc(nat) using le_imp_lesspoll[OF Card_csucc[OF Ord_nat]]
lt_csucc[OF Ord_nat] le_iff by auto
  {
    fix T assume T{is a topology} (⋃T){is compact in}T
    then have (⋃T){is compact of cardinal}nat{in}T using Compact_is_card_nat
by auto
    then have (⋃T){is compact of cardinal}csucc(nat){in}T using s compact_greater_card
Card_csucc[OF Ord_nat] by auto
    then have (⋃T){is lindeloef in}T unfolding IsLindeloef_def by auto
  }
  then have ∀T. T{is a topology} ⟶ ((⋃T){is compact in}T) ⟶ ((⋃T){is
lindeloef in}T) by auto
  from eq_spect_rev_imp_anti[OF this] lindeloef_spec2[OF assms(1)] compact_spectrum
    show thesis using assms(2) unfolding IsAntiLin_def IsAntiComp_def
by auto
qed
```

If the axiom of countable choice holds for subsets of the power set of the natural numbers, then there exists a topological space that is anti-lindeloef but no anti-compact.

```
theorem no_choice_imp_anti_lindeloef_is_anti_comp:
  assumes ({the axiom of} nat {choice holds for subsets}(Pow(nat)))
  shows ({one-point compactification of}Pow(nat)){is anti-lindeloef}
proof-
  have t:⋃({one-point compactification of}Pow(nat))={nat}∪nat using topology0.op_compact_t
    unfolding topology0_def using Pow_is_top by auto
  have {nat}≈1 using singleton_eqpoll_1 by auto
  then have {nat}≺nat using n_lesspoll_nat eq_lesspoll_trans by auto
moreover
  have s:nat≺csucc(nat) using lt_Card_imp_lesspoll[OF Card_csucc] lt_csucc[OF
Ord_nat] by auto
  ultimately have {nat}≺csucc(nat) using lesspoll_trans by blast
  with s have {nat}∪nat≺csucc(nat) using less_less_imp_un_less[OF _ _
InfCard_csucc[OF InfCard_nat]]
    by auto
```

```
  then have {nat}∪nat≲nat using Card_less_csucc_eq_le[OF Card_nat] by
auto
  with t have r:⋃({one-point compactification of}Pow(nat))≲nat by auto
  {
    fix A assume A:A∈Pow(⋃({one-point compactification of}Pow(nat)))
(⋃(({one-point compactification of}Pow(nat)){restricted to}A)){is lindeloef
in}(({one-point compactification of}Pow(nat)){restricted to}A)
    from A(1) have A⊆⋃({one-point compactification of}Pow(nat)) by auto
    with r have A≲nat using subset_imp_lepoll lepoll_trans by blast
    then have A{is in the spectrum of}(λT. ((⋃T){is lindeloef in}T))
using assms
      lindeloef_spec1 by auto
  }
  then show thesis unfolding IsAntiLin_def antiProperty_def by auto
qed

theorem op_comp_pow_nat_no_anti_comp:
  shows ¬(({one-point compactification of}Pow(nat)){is anti-compact})
proof
  let T=({one-point compactification of}Pow(nat)){restricted to}({nat}
∪ nat)
  assume antiComp:({one-point compactification of}Pow(nat)){is anti-compact}
  have ({nat} ∪ nat){is compact in}({one-point compactification of}Pow(nat))
    using topology0.compact_op[of Pow(nat)] Pow_is_top[of nat] unfold-
ing topology0_def
    by auto
  then have ({nat} ∪ nat){is compact in}T using compact_imp_compact_subspace
Compact_is_card_nat by auto
  moreover have ⋃T=(⋃({one-point compactification of}Pow(nat)))∩({nat}
∪ nat) unfolding RestrictedTo_def by auto
  then have ⋃T={nat} ∪ nat using topology0.op_compact_total unfolding
topology0_def
    using Pow_is_top by auto
  ultimately have (⋃T){is compact in}T by auto
  with antiComp have ({nat} ∪ nat){is in the spectrum of}(λT. (⋃T){is
compact in}T) unfolding IsAntiComp_def
    antiProperty_def using topology0.op_compact_total unfolding topology0_def
using Pow_is_top by auto
  then have Finite({nat} ∪ nat) using compact_spectrum by auto
  then have Finite(nat) using subset_Finite by auto
  then show False using nat_not_Finite by auto
qed
```

In coclusion, we reached another equivalence of this choice principle.

The axiom of countable choice holds for subsets of the power set of the natural numbers if and only if there exists a topological space which is anti-lindeloef but not anti-compact; this space can be chosen as the one-point compactification of the discrete topology on ℕ.

```
theorem acc_pow_nat_equiv1:
  shows ({the axiom of} nat {choice holds for subsets}(Pow(nat))) ⟷
(({one-point compactification of}Pow(nat)){is anti-lindeloef})
  using op_comp_pow_nat_no_anti_comp no_choice_imp_anti_lindeloef_is_anti_comp
  topology0.no_choice_imp_anti_lindeloef_is_anti_comp topology0.op_comp_is_top
  Pow_is_top[of nat] unfolding topology0_def by auto

theorem acc_pow_nat_equiv2:
  shows ({the axiom of} nat {choice holds for subsets}(Pow(nat))) ⟷
(∃T. T{is a topology}
  ∧ (T{is anti-lindeloef}) ∧ ¬(T{is anti-compact}))
  using op_comp_pow_nat_no_anti_comp no_choice_imp_anti_lindeloef_is_anti_comp
  topology0.no_choice_imp_anti_lindeloef_is_anti_comp topology0.op_comp_is_top
  Pow_is_top[of nat] unfolding topology0_def by auto
```

In the file `Topology_ZF_properties.thy`, it is proven that $\mathbb{N}$ is lindeloef if and only if the axiom of countable choice holds for subsets of $Pow(\mathbb{N})$. Now we check that, in ZF, this space is always anti-lindeloef.

```
theorem nat_anti_lindeloef:
  shows Pow(nat){is anti-lindeloef}
proof-
  {
   fix A assume A:A∈Pow(⋃Pow(nat)) (⋃(Pow(nat){restricted to}A)){is
lindeloef in}(Pow(nat){restricted to}A)
    from A(1) have A⊆nat by auto
    then have Pow(nat){restricted to}A=Pow(A) unfolding RestrictedTo_def
by blast
    with A(2) have lin:A{is lindeloef in}Pow(A) using subset_imp_lepoll
by auto
    {
      fix T assume T:T{is a topology} ⋃T≈A
      then have A≈⋃T using eqpoll_sym by auto
      then obtain f where f:f∈bij(A,⋃T) unfolding eqpoll_def by auto
      then have f∈surj(A,⋃T) unfolding bij_def by auto
      moreover then have IsContinuous(Pow(A),T,f) unfolding IsContinuous_def
        surj_def using func1_1_L3 by blast
      moreover have two_top_spaces0(Pow(A),T,f) unfolding two_top_spaces0_def
        using f T(1) Pow_is_top unfolding bij_def inj_def by auto
      ultimately have (⋃T){is lindeloef in}T using two_top_spaces0.cont_image_com
        lin unfolding IsLindeloef_def by auto
    }
    then have A{is in the spectrum of} (λT. ((⋃T){is lindeloef in}T))
unfolding Spec_def by auto
  }
  then show thesis unfolding IsAntiLin_def antiProperty_def by auto
qed
```

This result is interesting because depending on the different axioms we add to ZF, it means two different things:

- Every subspace of ℕ is Lindeloef.
- Only the compact subspaces of ℕ are Lindeloef.

Now, we could wonder if the class of compact spaces and the class of lindeloef spaces being equal is consistent in ZF. Let's find a topological space which is lindeloef and no compact without assuming any axiom of choice or any negation of one. This will prove that the class of lindeloef spaces and the class of compact spaces cannot be equal in any model of ZF.

```
theorem lord_nat:
  shows (LOrdTopology nat Le)={LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0}
proof-
  {
    fix U assume U:U⊆{LeftRayX(nat,Le,n). n∈nat} ∪{nat} U≠0
    {
      assume nat∈U
      with U have ⋃U=nat unfolding LeftRayX_def by auto
      then have ⋃U∈{LeftRayX(nat,Le,n). n∈nat} ∪{nat}∪{0} by auto
    }
    moreover
    {
      assume nat∉U
      with U have UU:U⊆{LeftRayX(nat,Le,n). n∈nat}∪{0} by auto
      {
        assume A:∃i. i∈nat∧ ⋃U⊆ LeftRayX(nat,Le,i)
        let M=μ i. i∈nat ∧ ⋃U⊆ LeftRayX(nat,Le,i)
        from A have M:M∈nat ⋃U⊆ LeftRayX(nat,Le,M) using LeastI[OF _
nat_into_Ord, where P=λi. i∈nat ∧ ⋃U⊆ LeftRayX(nat,Le,i)]
          by auto
        {
          fix y assume V:y∈LeftRayX(nat,Le,M)
          then have y:y∈nat unfolding LeftRayX_def by auto
          {
            assume ∀V∈U. y∉V
            then have ∀m∈{n∈nat. LeftRayX(nat,Le,n)∈U}. y∉LeftRayX(nat,Le,m)
using UU by auto
            then have ∀m∈{n∈nat. LeftRayX(nat,Le,n)∈U}. ⟨y,m⟩∉Le∨y=m
unfolding LeftRayX_def using y
              by auto
            then have RR:∀m∈{n∈nat. LeftRayX(nat,Le,n)∈U}. ⟨m,y⟩∈Le us-
ing Le_directs_nat(1) y unfolding IsLinOrder_def IsTotal_def by blast
            {
              fix rr V assume rr∈⋃U
              then obtain V where V:V∈U rr∈V by auto
              with UU obtain m where m:V=LeftRayX(nat,Le,m) m∈nat by
auto
              with V(1) RR have a:⟨m,y⟩∈Le by auto
              from V(2) m(1) have b:⟨rr,m⟩∈Le rr∈nat-{m} unfolding LeftRayX_def
by auto
```

```
            from a b(1) have ⟨rr,y⟩∈Le using Le_directs_nat(1) unfolding IsLinOrder_def
              trans_def by blast moreover
            {
              assume rr=y
              with a b have False using Le_directs_nat(1) unfolding IsLinOrder_def antisym_def by blast
            }
            ultimately have rr∈LeftRayX(nat,Le,y) unfolding LeftRayX_def using b(2) by auto
          }
          then have ⋃U⊆LeftRayX(nat,Le,y) by auto
          with y M(1) have ⟨M,y⟩∈Le using Least_le by auto
          with V have False unfolding LeftRayX_def using Le_directs_nat(1) unfolding IsLinOrder_def antisym_def by blast
        }
        then have y∈⋃U by auto
      }
      then have LeftRayX(nat,Le,M)⊆⋃U by auto
      with M(2) have ⋃U=LeftRayX(nat,Le,M) by auto
      with M(1) have ⋃U∈{LeftRayX(nat,Le,n). n∈nat} ∪{nat} by auto
    }
    moreover
    {
      assume ¬(∃i. i ∈ nat ∧ ⋃U ⊆ LeftRayX(nat, Le, i))
      then have A:∀i. i∈nat ⟶ ¬(⋃U ⊆ LeftRayX(nat, Le, i)) by auto
      {
        fix i assume i:i∈nat
        with A have AA:¬(⋃U ⊆ LeftRayX(nat, Le, i)) by auto
        {
          assume i∉⋃U
          then have ∀V∈U. i∉V by auto
          then have ∀m∈{n∈nat. LeftRayX(nat, Le, n)∈U}. i∉LeftRayX(nat, Le, m) by auto
          with i have ∀m∈{n∈nat. LeftRayX(nat, Le, n)∈U}. ⟨i,m⟩∉Le∨i=m unfolding LeftRayX_def by auto
          with i have ∀m∈{n∈nat. LeftRayX(nat, Le, n)∈U}. ¬(i≤m)∨i=m unfolding Le_def by auto
          then have ∀m∈{n∈nat. LeftRayX(nat, Le, n)∈U}. m<i∨m=i using not_le_iff_lt[OF nat_into_Ord[OF i]
            nat_into_Ord] by auto
          then have M:∀m∈{n∈nat. LeftRayX(nat, Le, n)∈U}. m≤i using le_iff nat_into_Ord[OF i] by auto
          {
            fix s assume s∈⋃U
            then obtain n where n:n∈nat s∈LeftRayX(nat, Le, n) LeftRayX(nat, Le, n)∈U
              using UU by auto
            with M have ni:n≤i by auto
```

```
              from n(2) have sn:s≤n s≠n unfolding LeftRayX_def by auto
              then have s≤i s≠i using le_trans[OF sn(1) ni] le_anti_sym[OF
sn(1)] ni by auto
              then have s∈LeftRayX(nat, Le, i) using i le_in_nat un-
folding LeftRayX_def by auto
            }
            with AA have False by auto
          }
          then have i∈⋃U by auto
        }
        then have nat⊆⋃U by auto
        then have ⋃U=nat using UU unfolding LeftRayX_def by auto
        then have ⋃U∈{LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0} by auto
      }
      ultimately have ⋃U∈{LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0} by
auto
    }
    ultimately have ⋃U∈{LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0} by auto
  }
  moreover
  {
    fix U assume U=0
    then have ⋃U∈{LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0} by auto
  }
  ultimately have ∀U. U⊆{LeftRayX(nat,Le,n). n∈nat} ∪{nat} ⟶ ⋃U∈{LeftRayX(nat,Le,n).
n∈nat} ∪{nat} ∪{0}
    by auto
  then have {LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0}={⋃U. U∈Pow({LeftRayX(nat,Le,n).
n∈nat} ∪{nat})} by blast
  then show thesis using LOrdtopology_ROrdtopology_are_topologies(2)[OF
Le_directs_nat(1)]
    unfolding IsAbaseFor_def by auto
qed

lemma countable_lord_nat:
  shows {LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0}≺csucc(nat)
proof-
  {
    fix e
    have {e}≈1 using singleton_eqpoll_1 by auto
    then have {e}≺nat using n_lesspoll_nat eq_lesspoll_trans by auto
moreover
    have s:nat≺csucc(nat) using lt_Card_imp_lesspoll[OF Card_csucc] lt_csucc[OF
Ord_nat] by auto
    ultimately have {e}≺csucc(nat) using lesspoll_trans by blast
  }
  then have {nat} ∪{0}≺csucc(nat) using less_less_imp_un_less[OF _ _
InfCard_csucc[OF InfCard_nat], of {nat} {0}]
    by auto moreover
```

```
  let FF={⟨n,LeftRayX(nat,Le,n)⟩. n∈nat}
  have ff:FF:nat→{LeftRayX(nat,Le,n). n∈nat} unfolding Pi_def domain_def
function_def by auto
  then have su:FF∈surj(nat,{LeftRayX(nat,Le,n). n∈nat}) unfolding surj_def
using apply_equality[
    OF _ ff] by auto
  then have {LeftRayX(nat,Le,n). n∈nat}≲nat using surj_fun_inv_2[OF su
lepoll_refl[of nat]] Ord_nat
    by auto
  then have {LeftRayX(nat,Le,n). n∈nat}≺csucc(nat) using Card_less_csucc_eq_le[OF
Card_nat] by auto
  ultimately have {LeftRayX(nat, Le, n) . n ∈ nat} ∪ ({nat} ∪ {0}) ≺
csucc(nat) using less_less_imp_un_less[OF _ _ InfCard_csucc[OF InfCard_nat]]
by auto
  moreover have {LeftRayX(nat, Le, n) . n ∈ nat} ∪ ({nat} ∪ {0})={LeftRayX(nat,
Le, n) . n ∈ nat} ∪ {nat} ∪ {0} by auto
  ultimately show thesis by auto
qed

corollary lindelof_lord_nat:
  shows nat{is lindeloef in}(LOrdTopology nat Le)
  unfolding IsLindeloef_def using countable_lord_nat lord_nat card_top_comp[OF
Card_csucc[OF Ord_nat]]
    union_lordtopology_rordtopology(1)[OF Le_directs_nat(1)] by auto

theorem not_comp_lord_nat:
  shows ¬(nat{is compact in}(LOrdTopology nat Le))
proof
  assume nat{is compact in}(LOrdTopology nat Le)
  with lord_nat have nat{is compact in}({LeftRayX(nat,Le,n). n∈nat} ∪{nat}
∪{0}) by auto
  then have ∀M∈Pow({LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0}). nat⊆⋃M
⟶ (∃N∈FinPow(M). nat⊆⋃N)
    unfolding IsCompact_def by auto moreover
  {
    fix n assume n:n∈nat
    then have n<succ(n) by auto
    then have ⟨n,succ(n)⟩∈Le n≠succ(n) using n nat_succ_iff by auto
    then have n∈LeftRayX(nat,Le,succ(n)) unfolding LeftRayX_def using
n by auto
    then have n∈⋃({LeftRayX(nat,Le,n). n∈nat}) using n nat_succ_iff
by auto
  }
  ultimately have ∃N∈FinPow({LeftRayX(nat,Le,n). n∈nat}). nat⊆⋃N by
blast
  then obtain N where N∈FinPow({LeftRayX(nat,Le,n). n∈nat}) nat⊆⋃N
by auto
  then have N:N⊆{LeftRayX(nat,Le,n). n∈nat} Finite(N) nat⊆⋃N unfold-
ing FinPow_def by auto
```

```
  let F={⟨n,LeftRayX(nat,Le,n)⟩. n∈{m∈nat. LeftRayX(nat,Le,m)∈N}}
  have ff:F:{m∈nat. LeftRayX(nat,Le,m)∈N} → N unfolding Pi_def function_def
by auto
  then have F∈surj({m∈nat. LeftRayX(nat,Le,m)∈N}, N) unfolding surj_def
using N(1) apply_equality[
    OF _ ff] by blast moreover
  {
    fix x y assume xyF:x∈{m∈nat. LeftRayX(nat,Le,m)∈N} y∈{m∈nat. LeftRayX(nat,Le,m)∈N}
Fx=Fy
    then have Fx=LeftRayX(nat,Le,x) Fy=LeftRayX(nat,Le,y) using apply_equality[
      OF _ ff] by auto
    with xyF(3) have lxy:LeftRayX(nat,Le,x)=LeftRayX(nat,Le,y) by auto
    {
      fix r assume r<x
      then have r≤x r≠x using leI by auto
      with xyF(1) have r∈LeftRayX(nat,Le,x) unfolding LeftRayX_def us-
ing le_in_nat by auto
      then have r∈LeftRayX(nat,Le,y) using lxy by auto
      then have r≤y r≠y unfolding LeftRayX_def by auto
      then have r<y using le_iff by auto
    }
    then have ∀r. r<x ⟶ r<y by auto
    then have r:¬(y<x) by auto
    {
      fix r assume r<y
      then have r≤y r≠y using leI by auto
      with xyF(2) have r∈LeftRayX(nat,Le,y) unfolding LeftRayX_def us-
ing le_in_nat by auto
      then have r∈LeftRayX(nat,Le,x) using lxy by auto
      then have r≤x r≠x unfolding LeftRayX_def by auto
      then have r<x using le_iff by auto
    }
    then have ¬(x<y) by auto
    with r have x=y using not_lt_iff_le[OF nat_into_Ord nat_into_Ord]
xyF(1,2)
      le_anti_sym by auto
  }
  then have F∈inj({m∈nat. LeftRayX(nat,Le,m)∈N}, N) unfolding inj_def
using ff by auto
  ultimately have F∈bij({m∈nat. LeftRayX(nat,Le,m)∈N}, N) unfolding bij_def
by auto
  then have {m∈nat. LeftRayX(nat,Le,m)∈N}≈ N unfolding eqpoll_def by
auto
  with N(2) have fin:Finite({m∈nat. LeftRayX(nat,Le,m)∈N}) using lepoll_Finite
eqpoll_imp_lepoll
    by auto
  from N(3) have N≠0 by auto
  then have nE:{m∈nat. LeftRayX(nat,Le,m)∈N}≠0 using N(1) by auto
  let M=Maximum(Le,{m∈nat. LeftRayX(nat,Le,m)∈N})
```

```
  have M:M∈nat LeftRayX(nat,Le,M)∈N ∀𝔯∈{m∈nat. LeftRayX(nat,Le,m)∈N}.
⟨𝔯,M⟩∈Le using fin linord_max_props(1,3)[OF Le_directs_nat(1) _ nE]
    unfolding FinPow_def by auto
  {
    fix V 𝔯 assume V:V∈N 𝔯∈V
    then obtain m where m:V=LeftRayX(nat,Le,m) LeftRayX(nat,Le,m)∈N m∈nat
using N(1) by auto
    with V(2) have xx:⟨𝔯,m⟩∈Le 𝔯≠m unfolding LeftRayX_def by auto
    from m(2,3) have m∈{m∈nat. LeftRayX(nat,Le,m)∈N} by auto
    then have mM:⟨m,M⟩∈Le using M(3) by auto
    with xx(1) have ⟨𝔯,M⟩∈Le using le_trans unfolding Le_def by auto
    moreover
    {
      assume 𝔯=M
      with xx mM have False using le_anti_sym by auto
    }
    ultimately have 𝔯∈LeftRayX(nat,Le,M) unfolding LeftRayX_def by auto
  }
  then have ⋃N⊆LeftRayX(nat,Le,M) by auto
  with M(2) have ⋃N=LeftRayX(nat,Le,M) by auto
  with N(3) have nat⊆LeftRayX(nat,Le,M) by auto
  moreover from M(1) have succ(M)∈nat using nat_succI by auto
  ultimately have succ(M)∈LeftRayX(nat,Le,M) by auto
  then have ⟨succ(M),M⟩∈Le unfolding LeftRayX_def by blast
  then show False by auto
qed
```

89.4 More Separation properties

In this section we study more separation properties.

89.5 Definitions

We start with a property that has already appeared in Topology_ZF_1b.thy.
A KC-space is a space where compact sets are closed.

```
definition
  IsKC (_ {is KC}) where
  T{is KC} ≡ ∀A∈Pow(⋃T). A{is compact in}T ⟶ A{is closed in}T
```

Another type of space is an US-space; those where sequences have at most one limit.

```
definition
  IsUS (_{is US}) where
  T{is US} ≡ ∀N x y. (N:nat→⋃T) ∧ NetConvTop(⟨N,Le⟩,x,T) ∧ NetConvTop(⟨N,Le⟩,y,T)
⟶ y=x
```

89.6 First results

The proof in Topology_ZF_1b.thy shows that a Hausdorff space is KC.

```
corollary(in topology0) T2_imp_KC:
  assumes T{is T₂}
  shows T{is KC}
proof-
  {
    fix A assume A{is compact in}T
    then have A{is closed in}T using in_t2_compact_is_cl assms by auto
  }
  then show thesis unfolding IsKC_def by auto
qed
```

From the spectrum of compactness, it follows that any KC-space is T_1.

```
lemma(in topology0) KC_imp_T1:
  assumes T{is KC}
  shows T{is T₁}
proof-
  {
    fix x assume A:x∈⋃T
    have Finite({x}) by auto
    then have {x}{is in the spectrum of}(λT. (⋃T){is compact in}T)
      using compact_spectrum by auto moreover
    have (T{restricted to}{x}){is a topology} using Top_1_L4 by auto
    moreover have ⋃(T{restricted to}{x})={x} using A unfolding RestrictedTo_def
by auto
    ultimately have com:{x}{is compact in}(T{restricted to}{x}) unfold-
ing Spec_def
      by auto
    then have {x}{is compact in}T using compact_subspace_imp_compact
A by auto
    then have {x}{is closed in}T using assms unfolding IsKC_def using
A by auto
  }
  then show thesis using T1_iff_singleton_closed by auto
qed
```

Even more, if a space is KC, then it is US. We already know that for T_2 spaces, any net or filter has at most one limit; and that this property is equivalent with T_2. The US property is much weaker because we don't know what happends with other nets that are not directed by the order on the natural numbers.

```
theorem(in topology0) KC_imp_US:
  assumes T{is KC}
  shows T{is US}
proof-
  {
```

```
fix N x y  assume A:N:nat→⋃T ⟨N,Le⟩→N x ⟨N,Le⟩→N y x≠y
have dir:Le directs nat using Le_directs_nat by auto moreover
from A(1) have dom:domain(N)=nat using func1_1_L1 by auto
moreover note A(1) ultimately have Net:⟨N,Le⟩{is a net on}⋃T unfolding IsNet_def
  by auto
from A(3) have y:y∈⋃T unfolding NetConverges_def[OF Net] by auto
from A(2) have x:x∈⋃T unfolding NetConverges_def[OF Net] by auto
from A(2) have o1:∀U∈Pow(⋃T). x∈int(U) ⟶ (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) unfolding NetConverges_def[OF Net]
  using dom by auto
{
  assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=y
  have {y}{is closed in}T using y T1_iff_singleton_closed KC_imp_T1 assms by auto
  then have o2:⋃T-{y}∈T unfolding IsClosed_def by auto
  then have int(⋃T-{y})=⋃T-{y} using Top_2_L3 by auto
  with A(4) x have o3:x∈int(⋃T-{y}) by auto
  from o2 have ⋃T-{y}∈Pow(⋃T) by auto
  with o1 o3 obtain r where r:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈⋃T-{y} by auto
  from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=y by auto
  from dir r(1) n(1) obtain z where ⟨r,z⟩∈Le⟨n,z⟩∈Lez∈nat unfolding IsDirectedSet_def by auto
  with r(2) n(2) have Nz∈⋃T-{y} Nz=y by auto
  then have False by auto
}
then have reg:∀n∈nat. ∃m∈nat. Nm≠y ∧ ⟨n,m⟩∈Le by auto
let NN={⟨n,N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)⟩. n∈nat}
{
  fix x z assume A1:⟨x, z⟩ ∈ NN
  {
    fix y' assume A2:⟨x,y'⟩∈NN
    with A1 have z=y' by auto
  }
  then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
}
then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
moreover
{
  fix n assume as:n∈nat
  with reg obtain m where Nm≠y ∧ ⟨n,m⟩∈Le m∈nat by auto
  then have LI:N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)≠y ⟨n,μ i. Ni≠y ∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y ∧ ⟨n,m⟩∈Le m]
    nat_into_Ord[of m] by auto
  then have (μ i. Ni≠y ∧ ⟨n,i⟩∈Le)∈nat by auto
  then have N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)∈⋃T using apply_type[OF A(1)] by auto
  with as have ⟨n,N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃T by auto
```

```
    }
    then have NN∈Pow(nat×⋃T) by auto
    ultimately have NFun:NN:nat→⋃T unfolding Pi_def function_def domain_def
by auto
    {
      fix n assume as:n∈nat
      with reg obtain m where Nm≠y ∧ ⟨n,m⟩∈Le m∈nat by auto
      then have LI:N(μ i. Ni≠y ∧ ⟨n,i⟩∈Le)≠y ⟨n,μ i. Ni≠y ∧ ⟨n,i⟩∈Le⟩∈Le
using LeastI[of λm. Nm≠y ∧ ⟨n,m⟩∈Le m]
        nat_into_Ord[of m] by auto
      then have NNn≠y using apply_equality[OF _ NFun] by auto
    }
    then have noy:∀n∈nat. NNn≠y by auto
    have dom2:domain(NN)=nat by auto
    then have net2:⟨NN,Le⟩{is a net on}⋃T unfolding IsNet_def using NFun
dir by auto
    {
      fix U assume U∈Pow(⋃T) x∈int(U)
      then have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) using o1 by auto
      then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U by auto
      {
        fix s assume AA:⟨r,s⟩∈Le
        with reg obtain m where Nm≠y ⟨s,m⟩∈Le by auto
        then have ⟨s,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y
∧ ⟨s,m⟩∈Le m]
          nat_into_Ord by auto
        with AA have ⟨r,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by auto
        with r_def(2) have N(μ i. Ni≠y ∧ ⟨s,i⟩∈Le)∈U by blast
        then have NNs∈U using apply_equality[OF _ NFun] AA by auto
      }
      then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
      with r_def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
    }
    then have conv2:⟨NN,Le⟩→N x unfolding NetConverges_def[OF net2] us-
ing x dom2 by auto
    let A={x}∪NNnat
    {
      fix M assume Acov:A⊆⋃M M⊆T
      then have x∈⋃M by auto
      then obtain U where U:x∈U U∈M by auto
      with Acov(2) have UT:U∈T by auto
      then have U=int(U) using Top_2_L3 by auto
      with U(1) have x∈int(U) by auto
      with conv2 obtain r where rr:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U
        unfolding NetConverges_def[OF net2] using dom2 UT by auto
      have NresFun:restrict(NN,{n∈nat. ⟨n,r⟩∈Le}):{n∈nat. ⟨n,r⟩∈Le}→⋃T
using restrict_fun
        [OF NFun, of {n∈nat. ⟨n,r⟩∈Le}] by auto
      then have restrict(NN,{n∈nat. ⟨n,r⟩∈Le})∈surj({n∈nat. ⟨n,r⟩∈Le},range(restrict(NN,{n
```

```
⟨n,r⟩∈Le}))
        using fun_is_surj by auto moreover
      have {n∈nat. ⟨n,r⟩∈Le}⊆nat by auto
      then have {n∈nat. ⟨n,r⟩∈Le}≲nat using subset_imp_lepoll by auto
      ultimately have range(restrict(NN,{n∈nat. ⟨n,r⟩∈Le}))≲{n∈nat. ⟨n,r⟩∈Le}
using surj_fun_inv_2 by auto
      moreover
      have {n∈nat. ⟨n,0⟩∈Le}={0} by auto
      then have Finite({n∈nat. ⟨n,0⟩∈Le}) by auto moreover
      {
        fix j assume as:j∈nat Finite({n∈nat. ⟨n,j⟩∈Le})
        {
          fix t assume t∈{n∈nat. ⟨n,succ(j)⟩∈Le}
          then have t∈nat ⟨t,succ(j)⟩∈Le by auto
          then have t≤succ(j) by auto
          then have t⊆succ(j) using le_imp_subset by auto
          then have t⊆j ∪{j} using succ_explained by auto
          then have j∈t∨t⊆j by auto
          then have j∈t∨t≤j using subset_imp_le ‹t∈nat› ‹j∈nat› nat_into_Ord
by auto
          then have j ∪{j}⊆t∨t≤j using ‹t∈nat› ‹j∈nat› nat_into_Ord
unfolding Ord_def
            Transset_def by auto
          then have succ(j)⊆t∨t≤j using succ_explained by auto
          with ‹t⊆succ(j)› have t=succ(j)∨t≤j by auto
          with ‹t∈nat› ‹j∈nat› have t∈{n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}
by auto
        }
        then have {n∈nat. ⟨n,succ(j)⟩∈Le} ⊆{n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}
by auto
        moreover have Finite({n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}) using as(2)
Finite_cons
          by auto
        ultimately have Finite({n∈nat. ⟨n,succ(j)⟩∈Le}) using subset_Finite
by auto
      }
      then have ∀j∈nat. Finite({n∈nat. ⟨n,j⟩∈Le}) ⟶ Finite({n∈nat.
⟨n,succ(j)⟩∈Le})
        by auto
      ultimately have Finite(range(restrict(NN, {n ∈ nat . ⟨n, r⟩ ∈ Le})))
        using lepoll_Finite[of range(restrict(NN, {n ∈ nat . ⟨n, r⟩ ∈
Le}))
          {n ∈ nat . ⟨n, r⟩ ∈ Le}] ind_on_nat[OF ‹r∈nat›, where P=λt.
Finite({n∈nat. ⟨n,t⟩∈Le})] by auto
      then have Finite((restrict(NN, {n ∈ nat . ⟨n, r⟩ ∈ Le})){n ∈ nat
. ⟨n, r⟩ ∈ Le}) using range_image_domain[OF NresFun]
        by auto
      then have Finite(NN{n ∈ nat . ⟨n, r⟩ ∈ Le}) using restrict_image
by auto
```

```
      then have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is in the spectrum of}(λT.
(⋃T){is compact in}T) using compact_spectrum by auto
      moreover have ⋃(T{restricted to}NN{n ∈ nat . ⟨n, r⟩ ∈ Le})=⋃T∩NN{n
∈ nat . ⟨n, r⟩ ∈ Le}
        unfolding RestrictedTo_def by auto moreover
      have ⋃T∩NN{n ∈ nat . ⟨n, r⟩ ∈ Le}=NN{n ∈ nat . ⟨n, r⟩ ∈ Le}
        using func1_1_L6(2)[OF NFun] by blast
      moreover have (T{restricted to}NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is a
topology}
        using Top_1_L4 by auto
      ultimately have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is compact in}(T{restricted
to}NN{n ∈ nat . ⟨n, r⟩ ∈ Le})
        unfolding Spec_def by force
      then have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is compact in}T using compact_subspace_imp_comp
by auto
      moreover from Acov(1) have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le})⊆⋃M by
auto
      moreover note Acov(2) ultimately
      obtain 𝔑 where 𝔑:𝔑∈FinPow(M) (NN{n ∈ nat . ⟨n, r⟩ ∈ Le})⊆⋃𝔑
        unfolding IsCompact_def by blast
      from 𝔑(1) have 𝔑 ∪{U}∈FinPow(M) using U(2) unfolding FinPow_def
by auto moreover
      {
        fix s assume s:s∈A s∉U
        with U(1) have s∈NNnat by auto
        then have s∈{NNn. n∈nat} using func_imagedef[OF NFun] by auto
        then obtain n where n:n∈nat s=NNn by auto
        {
          assume ⟨r,n⟩∈Le
          with rr have NNn∈U by auto
          with n(2) s(2) have False by auto
        }
        then have ⟨r,n⟩∉Le by auto
        with rr(1) n(1) have ¬(r≤n) by auto
        then have n≤r using Ord_linear_le[where thesis=⟨n,r⟩∈Le] nat_into_Ord[OF
rr(1)]
          nat_into_Ord[OF n(1)] by auto
        with rr(1) n(1) have ⟨n,r⟩∈Le by auto
        with n(2) have s∈{NNt. t∈{n∈nat. ⟨n,r⟩∈Le}} by auto moreover
        have {n∈nat. ⟨n,r⟩∈Le}⊆nat by auto
        ultimately have s∈NN{n∈nat. ⟨n,r⟩∈Le} using func_imagedef[OF NFun]
          by auto
        with 𝔑(2) have s∈⋃𝔑 by auto
      }
      then have A⊆⋃𝔑 ∪ U by auto
      then have A⊆⋃(𝔑 ∪ {U}) by auto ultimately
      have ∃𝔑∈FinPow(M). A⊆⋃𝔑 by auto
    }
    then have ∀M∈Pow(T). A⊆⋃M ⟶ (∃𝔑∈FinPow(M). A⊆⋃𝔑) by auto more-
```

```
over
    have A⊆⋃T using func1_1_L6(2)[OF NFun] x by blast ultimately
    have A{is compact in}T unfolding IsCompact_def by auto
    with assms have A{is closed in}T unfolding IsKC_def IsCompact_def
by auto
    then have ⋃T-A∈T unfolding IsClosed_def by auto
    then have ⋃T-A=int(⋃T-A) using Top_2_L3 by auto moreover
    {
      assume y∈A
      with A(4) have y∈NNnat by auto
      then have y∈{NNn. n∈nat} using func_imagedef[OF NFun] by auto
      then obtain n where n∈natNNn=y by auto
      with noy have False by auto
    }
    with y have y∈⋃T-A by force ultimately
    have y∈int(⋃T-A) ⋃T-A∈Pow(⋃T) by auto moreover
    have (∀U∈Pow(⋃T).  y ∈ int(U) ⟶ (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le
⟶ N  m ∈ U))
      using A(3) dom unfolding NetConverges_def[OF Net] by auto
    ultimately have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ ⋃T-A by
blast
    then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈⋃T-A by
auto
    {
        fix s assume AA:⟨r,s⟩∈Le
        with reg obtain m where Nm≠y ⟨s,m⟩∈Le by auto
        then have ⟨s,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠y
∧ ⟨s,m⟩∈Le m]
          nat_into_Ord by auto
        with AA have ⟨r,μ i. Ni≠y ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by auto
        with r_def(2) have N(μ i. Ni≠y ∧ ⟨s,i⟩∈Le)∈⋃T-A by force
        then have NNs∈⋃T-A using apply_equality[OF _ NFun] AA by auto
        moreover have NNs∈{NNt. t∈nat} using AA by auto
        then have NNs∈NNnat using func_imagedef[OF NFun] by auto
        then have NNs∈A by auto
        ultimately have False by auto
      }
      moreover have r⊆succ(r) using succ_explained by auto
      then have r≤succ(r) using subset_imp_le nat_into_Ord ‹r∈nat› nat_succI
        by auto
      then have ⟨r,succ(r)⟩∈Le using ‹r∈nat› nat_succI by auto
      ultimately have False by auto
    }
    then have ∀N x y. (N:nat→⋃T) ∧ (⟨N, Le⟩ →_N x {in} T) ∧ (⟨N, Le⟩
→_N y {in} T)
      ⟶ x=y by auto
  then show thesis unfolding IsUS_def by auto
qed
```

US spaces are also T_1.

```
theorem (in topology0) US_imp_T1:
  assumes T{is US}
  shows T{is T₁}
proof-
  {
    fix x assume x:x∈⋃T
    then have {x}⊆⋃T by auto
    {
      fix y assume y:y≠x y∈cl({x})
      then have r:∀U∈T. y∈U ⟶ x∈U using cl_inter_neigh[OF ‹{x}⊆⋃T›]
by auto
      let N=ConstantFunction(nat,x)
      have fun:N:nat→⋃T using x func1_3_L1 by auto
      then have dom:domain(N)=nat using func1_1_L1 by auto
      with fun have Net:⟨N,Le⟩{is a net on}⋃T using Le_directs_nat un-
folding IsNet_def
        by auto
      {
        fix U assume U∈Pow(⋃T) x∈int(U)
        then have x∈U using Top_2_L1 by auto
        then have ∀n∈nat. Nn∈U using func1_3_L2 by auto
        then have ∀n∈nat. ⟨0,n⟩∈Le ⟶Nn∈U by auto
        then have ∃r∈nat. ∀n∈nat. ⟨r,n⟩∈Le ⟶Nn∈U by auto
      }
      then have ⟨N,Le⟩ →N x unfolding NetConverges_def[OF Net] using
x dom by auto moreover
      {
        fix U assume U∈Pow(⋃T) y∈int(U)
        then have x∈int(U) using r Top_2_L2 by auto
        then have x∈U using Top_2_L1 by auto
        then have ∀n∈nat. Nn∈U using func1_3_L2 by auto
        then have ∀n∈nat. ⟨0,n⟩∈Le ⟶Nn∈U by auto
        then have ∃r∈nat. ∀n∈nat. ⟨r,n⟩∈Le ⟶Nn∈U by auto
      }
      then have ⟨N,Le⟩ →N y unfolding NetConverges_def[OF Net] using
y(2) dom
        Top_3_L11(1)[OF ‹{x}⊆⋃T›] by auto
      ultimately have x=y using assms unfolding IsUS_def using fun by
auto
      with y(1) have False by auto
    }
    then have cl({x})⊆{x} by auto
    then have cl({x})={x} using cl_contains_set[OF ‹{x}⊆⋃T›] by auto
    then have {x}{is closed in}T using Top_3_L8 x by auto
  }
  then show thesis using T1_iff_singleton_closed by auto
qed
```

89.7 Counter-examples

We need to find counter-examples that prove that this properties are new ones.

We know that $T_2 \Rightarrow loc.T_2 \Rightarrow$ anti-hyperconnected $\Rightarrow T_1$ and $T_2 \Rightarrow KC \Rightarrow US \Rightarrow T_1$. The question is: What is the relation between KC or US and, $loc.T_2$ or anti-hyperconnected?

In the file `Topology_ZF_properties_2.thy` we built a topological space which is locally-T_2 but no T_2. It happends actually that this space is not even US given the appropiate topology `T`.

```
lemma (in topology0) locT2_not_US_1:
  assumes {m}∉T {m}{is closed in}T ∃N∈nat→⋃T. (⟨N,Le⟩→N m) ∧ m∉Nnat

  shows ∃N∈nat→⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}). (⟨N,Le⟩→N
⋃T {in} (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))
    ∧ (⟨N,Le⟩→N m {in} (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))
proof-
  from assms(3) obtain N where N:N:nat→⋃T ⟨N,Le⟩→N m m∉Nnat by auto
  have ⋃T⊆⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) using assms(2)
union_doublepoint_top
    by auto
  with N(1) have fun:N:nat→⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
using func1_1_L1B by auto
  then have dom:domain(N)=nat using func1_1_L1 by auto
  with fun have Net:⟨N,Le⟩{is a net on}⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T.
m∈V}×T}) unfolding
    IsNet_def using Le_directs_nat by auto
  from N(1) dom have Net2:⟨N,Le⟩{is a net on}⋃T unfolding IsNet_def us-
ing Le_directs_nat by auto
  from N(2) have R:∀U∈Pow(⋃T). m∈int(U) ⟶(∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le
⟶ Ns∈U)
    unfolding NetConverges_def[OF Net2] using dom by auto
  {
    fix U assume U:U∈Pow(⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))
m∈Interior(U,T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
    let I=Interior(U,T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
    have I∈T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} using topology0.Top_2_L2
assms(2) doble_point_top unfolding topology0_def by blast
    then have (⋃T)∩I∈(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){restricted
to}⋃T unfolding RestrictedTo_def by blast
    then have (⋃T)∩I∈T using open_subspace_double_point(1) assms(2)
by auto moreover
    then have int((⋃T)∩I)=(⋃T)∩I using Top_2_L3 by auto
    with U(2) assms(2) have m∈int((⋃T)∩I) unfolding IsClosed_def by
auto
    moreover note R ultimately have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈(⋃T)∩I
by blast
```

```
    then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈I by blast
    then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using topology0.Top_2_L1[of
T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}U] doble_point_top assms(2)
      unfolding topology0_def by auto
  }
  then have ∀U∈Pow(⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})). m∈Interior(U,T∪{(U-{m})
⟨U,W⟩∈{V∈T. m∈V}×T}) ⟶(∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) by auto
  moreover have tt:topology0(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
using doble_point_top[OF assms(2)] unfolding topology0_def.
  have m∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) using assms(2)
union_doublepoint_top unfolding IsClosed_def by auto ultimately
  have con1:(⟨N,Le⟩→N m {in} (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))
unfolding topology0.NetConverges_def[OF tt Net]
    using dom by auto
  {
    fix U assume U:U∈Pow(⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))
⋃T∈Interior(U,T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
    let I=Interior(U,T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})
    have I∈T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T} using topology0.Top_2_L2
assms(2) doble_point_top unfolding topology0_def by blast
    with U(2) mem_not_refl have I∈{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}
by auto
    then obtain V W where VW:I=(V-{m})∪{⋃T}∪W W∈T V∈T m∈V by auto
    from VW(3,4) have m∈int(V) using Top_2_L3 by auto moreover
    have V∈Pow(⋃T) using VW(3) by auto moreover
    note R ultimately
    have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈V by blast moreover
    from N(3) have ∀s∈nat. Ns≠m using func_imagedef[OF N(1)] by auto
ultimately
    have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈V-{m} by blast
    then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈I using VW(1) by auto
    then have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U using topology0.Top_2_L1
assms(2) doble_point_top unfolding topology0_def by blast
  }
  then have ∀U∈Pow(⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T})). ⋃T∈Interior(U,T∪{(U-{m
⟨U,W⟩∈{V∈T. m∈V}×T}) ⟶ (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U) by auto
  moreover have ⋃T∈⋃(T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}) us-
ing assms(2) union_doublepoint_top by auto ultimately
  have (⟨N,Le⟩→N ⋃T {in} (T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}))
unfolding topology0.NetConverges_def[OF tt Net]
    using dom by auto
  with con1 fun show thesis by auto
qed

corollary (in topology0) locT2_not_US_2:
  assumes {m}∉T {m}{is closed in}T ∃N∈nat→⋃T. (⟨N,Le⟩→N m) ∧ m∉Nnat
  shows ¬((T∪{(U-{m})∪{⋃T}∪W. ⟨U,W⟩∈{V∈T. m∈V}×T}){is US})
proof-
  have m≠⋃T using assms(2) mem_not_refl unfolding IsClosed_def by auto
```

```
  then show thesis using locT2_not_US_1 assms unfolding IsUS_def by blast
qed
```

In particular, we also know that a locally-T_2 space doesn't need to be KC; since KC⇒US. Also we know that anti-hyperconnected spaces don't need to be KC or US, since locally-T_2 ⇒anti-hyperconnected.

Let's find a KC space that is not T_2, an US space which is not KC and a T_1 space which is not US.

First, let's prove some lemmas about what relation is there between this properties under the influence of other ones. This will help us to find counter-examples.

Anti-compactness ereases the differences between several properties.

```
lemma (in topology0) anticompact_KC_equiv_T1:
  assumes T{is anti-compact}
  shows T{is KC}⟷T{is T1}
proof
  assume T{is KC}
  then show T{is T1} using KC_imp_T1 by auto
next
  assume AS:T{is T1}
  {
    fix A assume A:A{is compact in}T A∈Pow(⋃T)
    then have A{is compact in}(T{restricted to}A) A∈Pow(⋃T) using compact_imp_compact_subs
      Compact_is_card_nat by auto
    moreover then have ⋃(T{restricted to}A)=A unfolding RestrictedTo_def
by auto
    ultimately have (⋃(T{restricted to}A)){is compact in}(T{restricted
to}A) A∈Pow(⋃T) by auto
    with assms have Finite(A) unfolding IsAntiComp_def antiProperty_def
using compact_spectrum by auto
    then obtain n where n∈nat A≈n unfolding Finite_def by auto
    then have A≺nat using eq_lesspoll_trans n_lesspoll_nat by auto more-
over
    have ⋃T-(⋃T-A)=A using A(2) by auto
    ultimately have ⋃T-(⋃T-A)≺nat by auto
    then have ⋃T-A∈CoFinite ⋃T unfolding Cofinite_def CoCardinal_def
by auto
    then have ⋃T-A∈T using AS T1_cocardinal_coarser by auto
    with A(2) have A{is closed in}T unfolding IsClosed_def by auto
  }
  then show T{is KC} unfolding IsKC_def by auto
qed
```

Then if we find an anti-compact and T_1 but no T_2 space, there is a counter-example for $KC \Rightarrow T_2$. A counter-example for US doesn't need to be KC mustn't be anti-compact.

The cocountable topology on csucc(nat) is such a topology.

The cocountable topology on $\mathbb{N}^+$ is hyperconnected.

```
lemma cocountable_in_csucc_nat_HConn:
  shows (CoCountable csucc(nat)){is hyperconnected}
proof-
  {
    fix U V assume as:U∈(CoCountable csucc(nat))V∈(CoCountable csucc(nat))U∩V=0
    then have csucc(nat)-U≺csucc(nat)∨U=0csucc(nat)-V≺csucc(nat)∨V=0
      unfolding Cocountable_def CoCardinal_def by auto
    then have (csucc(nat)-U)∪(csucc(nat)-V)≺csucc(nat)∨U=0∨V=0 using
less_less_imp_un_less[
      OF _ _ InfCard_csucc[OF InfCard_nat]] by auto moreover
    {
      assume (csucc(nat)-U)∪(csucc(nat)-V)≺csucc(nat) moreover
      have (csucc(nat)-U)∪(csucc(nat)-V)=csucc(nat)-U∩V by auto
      with as(3) have (csucc(nat)-U)∪(csucc(nat)-V)=csucc(nat) by auto
      ultimately have csucc(nat)≺csucc(nat) by auto
      then have False by auto
    }
    ultimately have U=0∨V=0 by auto
  }
  then show (CoCountable csucc(nat)){is hyperconnected} unfolding IsHConnected_def
by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ is not anti-hyperconnected.

```
corollary cocountable_in_csucc_nat_notAntiHConn:
  shows ¬((CoCountable csucc(nat)){is anti-}IsHConnected)
proof
  assume as:(CoCountable csucc(nat)){is anti-}IsHConnected
  have (CoCountable csucc(nat)){is hyperconnected} using cocountable_in_csucc_nat_HConn
by auto moreover
  have csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  then have uni:⋃(CoCountable csucc(nat))=csucc(nat) using union_cocardinal
unfolding Cocountable_def by auto
  have ∀A∈(CoCountable csucc(nat)). A⊆⋃(CoCountable csucc(nat)) by
fast
  with uni have ∀A∈(CoCountable csucc(nat)). A⊆csucc(nat) by auto
  then have ∀A∈(CoCountable csucc(nat)). csucc(nat)∩A=A by auto
  ultimately have ((CoCountable csucc(nat)){restricted to}csucc(nat)){is
hyperconnected}
    unfolding RestrictedTo_def by auto
  with as have (csucc(nat)){is in the spectrum of}IsHConnected unfold-
ing antiProperty_def
    using uni by auto
  then have csucc(nat)≲1 using HConn_spectrum by auto
  then have csucc(nat)≺nat using n_lesspoll_nat lesspoll_trans1 by auto
  then show False using lt_csucc[OF Ord_nat] lt_Card_imp_lesspoll[OF
```

```
Card_csucc[OF Ord_nat]]
    lesspoll_trans by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ is not T_2.

```
theorem cocountable_in_csucc_nat_noT2:
  shows ¬(CoCountable csucc(nat)){is T₂}
proof
  assume (CoCountable csucc(nat)){is T₂}
  then have antiHC:(CoCountable csucc(nat)){is anti-}IsHConnected
    using topology0.T2_imp_anti_HConn[OF topology0_CoCardinal[OF InfCard_csucc[OF
InfCard_nat]]]
    unfolding Cocountable_def by auto
  then show False using cocountable_in_csucc_nat_notAntiHConn by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ is T_1.

```
theorem cocountable_in_csucc_nat_T1:
  shows (CoCountable csucc(nat)){is T₁}
  using cocardinal_is_T1[OF InfCard_csucc[OF InfCard_nat]] unfolding Cocountable_def
by auto
```

The cocountable topology on $\mathbb{N}^+$ is anti-compact.

```
theorem cocountable_in_csucc_nat_antiCompact:
  shows (CoCountable csucc(nat)){is anti-compact}
proof-
  have noE:csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  {
    fix A assume as:A⊆⋃(CoCountable csucc(nat)) (⋃((CoCountable csucc(nat)){restricted
to}A)){is compact in}((CoCountable csucc(nat)){restricted to}A)
    from as(1) have ass:A⊆csucc(nat) using union_cocardinal[OF noE] un-
folding Cocountable_def by auto
    have ((CoCountable csucc(nat)){restricted to}A)=CoCountable (A∩csucc(nat))
using subspace_cocardinal
      unfolding Cocountable_def by auto moreover
    from ass have A∩csucc(nat)=A by auto
    ultimately have ((CoCountable csucc(nat)){restricted to}A)=CoCountable
A by auto
    with as(2) have comp:(⋃(CoCountable A)){is compact in}(CoCountable
A) by auto
    {
      assume as2:A≺csucc(nat) moreover
      {
        fix t assume t:t∈A
        have A-{t}⊆A by auto
        then have A-{t}≲A using subset_imp_lepoll by auto
        with as2 have A-{t}≺csucc(nat) using lesspoll_trans1 by auto
moreover note noE
```

```
        ultimately have (A-{t}){is closed in}(CoCountable A) using closed_sets_cocardinal[o
csucc(nat)
          A-{t}A] unfolding Cocountable_def by auto
        then have A-(A-{t})∈(CoCountable A) unfolding IsClosed_def us-
ing union_cocardinal[OF noE, of A]
          unfolding Cocountable_def by auto moreover
        from t have A-(A-{t})={t} by auto ultimately
        have {t}∈(CoCountable A) by auto
      }
      then have r:∀t∈A. {t}∈(CoCountable A) by auto
      {
        fix U assume U:U∈Pow(A)
        {
          fix t assume t∈U
          with U r have t∈{t}{t}⊆U{t}∈(CoCountable A) by auto
          then have ∃V∈(CoCountable A). t∈V ∧ V⊆U by auto
        }
        then have U∈(CoCountable A) using topology0.open_neigh_open[OF
topology0_CoCardinal[
          OF InfCard_csucc[OF InfCard_nat]]] unfolding Cocountable_def
by auto
      }
      then have Pow(A)⊆(CoCountable A) by auto moreover
      {
        fix B assume B∈(CoCountable A)
        then have B∈Pow(⋃(CoCountable A)) by auto
        then have B∈Pow(A) using union_cocardinal[OF noE] unfolding Cocountable_def
by auto
      }
      ultimately have p:Pow(A)=(CoCountable A) by auto
      then have (CoCountable A){is anti-compact} using pow_anti_compact[of
A] by auto moreover
      from p have ⋃(CoCountable A)=⋃Pow(A) by auto
      then have tot:⋃(CoCountable A)=A by auto
      from comp have (⋃((CoCountable A){restricted to}(⋃(CoCountable
A)))){is compact in}((CoCountable A){restricted to}(⋃(CoCountable A)))
using compact_imp_compact_subspace
        Compact_is_card_nat tot unfolding RestrictedTo_def by auto
      ultimately have A{is in the spectrum of}(λT. (⋃T){is compact in}T)
        using comp tot unfolding IsAntiComp_def antiProperty_def by auto
    }
    moreover
    {
      assume as1:¬(A≺csucc(nat))
      from ass have A≲csucc(nat) using subset_imp_lepoll by auto
      with as1 have A≈csucc(nat) using lepoll_iff_leqpoll by auto
      then have csucc(nat)≈A using eqpoll_sym by auto
      then have nat≺A using lesspoll_eq_trans lt_csucc[OF Ord_nat]
        lt_Card_imp_lesspoll[OF Card_csucc[OF Ord_nat]] by auto
```

```
      then have nat≲A using lepoll_iff_leqpoll by auto
      then obtain f where f∈inj(nat,A) unfolding lepoll_def by auto
moreover
      then have fun:f:nat→A unfolding inj_def by auto
      then have f∈surj(nat,range(f)) using fun_is_surj by auto
      ultimately have f∈bij(nat,range(f)) unfolding bij_def inj_def surj_def
by auto
      then have nat≈range(f) unfolding eqpoll_def by auto
      then have e:range(f)≈nat using eqpoll_sym by auto
      then have as2:range(f)≺csucc(nat) using lt_Card_imp_lesspoll[OF
Card_csucc[OF Ord_nat]]
        lt_csucc[OF Ord_nat] eq_lesspoll_trans by auto
      then have range(f){is closed in}(CoCountable A) using closed_sets_cocardinal[of
csucc(nat)
          range(f)A] unfolding Cocountable_def using func1_1_L5B[OF fun]
noE by auto
      then have (A∩range(f)){is compact in}(CoCountable A) using compact_closed
union_cocardinal[OF noE, of A]
        comp Compact_is_card_nat unfolding Cocountable_def by auto
      moreover have int:A∩range(f)=range(f)range(f)∩A=range(f) using
func1_1_L5B[OF fun] by auto
      ultimately have range(f){is compact in}(CoCountable A) by auto
      then have range(f){is compact in}((CoCountable A){restricted to}range(f))
using compact_imp_compact_subspace
        Compact_is_card_nat by auto
      moreover have ((CoCountable A){restricted to}range(f))=CoCountable
(range(f)∩A)
        using subspace_cocardinal unfolding Cocountable_def by auto
      with int(2) have ((CoCountable A){restricted to}range(f))=CoCountable
range(f) by auto
      ultimately have comp2:range(f){is compact in}(CoCountable range(f))
by auto
      {
        fix t assume t:t∈range(f)
        have range(f)-{t}⊆range(f) by auto
        then have range(f)-{t}≲range(f) using subset_imp_lepoll by auto
        with as2 have range(f)-{t}≺csucc(nat) using lesspoll_trans1 by
auto moreover note noE
        ultimately have (range(f)-{t}){is closed in}(CoCountable range(f))
using closed_sets_cocardinal[of csucc(nat)
          range(f)-{t}range(f)] unfolding Cocountable_def by auto
        then have range(f)-(range(f)-{t})∈(CoCountable range(f)) un-
folding IsClosed_def using union_cocardinal[OF noE, of range(f)]
          unfolding Cocountable_def by auto moreover
        from t have range(f)-(range(f)-{t})={t} by auto ultimately
        have {t}∈(CoCountable range(f)) by auto
      }
      then have r:∀t∈range(f). {t}∈(CoCountable range(f)) by auto
      {
```

```
        fix U assume U:U∈Pow(range(f))
        {
          fix t assume t∈U
          with U r have t∈{t}{t}⊆U{t}∈(CoCountable range(f)) by auto
          then have ∃V∈(CoCountable range(f)). t∈V ∧ V⊆U by auto
        }
        then have U∈(CoCountable range(f)) using topology0.open_neigh_open[OF
topology0_CoCardinal[
          OF InfCard_csucc[OF InfCard_nat]]] unfolding Cocountable_def
by auto
      }
      then have Pow(range(f))⊆(CoCountable range(f)) by auto moreover
      {
        fix B assume B∈(CoCountable range(f))
        then have B∈Pow(⋃(CoCountable range(f))) by auto
        then have B∈Pow(range(f)) using union_cocardinal[OF noE] un-
folding Cocountable_def by auto
      }
      ultimately have p:Pow(range(f))=(CoCountable range(f)) by blast
      then have (CoCountable range(f)){is anti-compact} using pow_anti_compact[of
range(f)] by auto moreover
      from p have ⋃(CoCountable range(f))=⋃Pow(range(f)) by auto
      then have tot:⋃(CoCountable range(f))=range(f) by auto
      from comp2 have (⋃((CoCountable range(f)){restricted to}(⋃(CoCountable
range(f))))){is compact in}((CoCountable range(f)){restricted to}(⋃(CoCountable
range(f)))) using compact_imp_compact_subspace
        Compact_is_card_nat tot unfolding RestrictedTo_def by auto
      ultimately have range(f){is in the spectrum of}(λT. (⋃T){is compact
in}T)
        using comp tot unfolding IsAntiComp_def antiProperty_def by auto
      then have Finite(range(f)) using compact_spectrum by auto
      then have Finite(nat) using e eqpoll_imp_Finite_iff by auto
      then have False using nat_not_Finite by auto
    }
    ultimately have A{is in the spectrum of}(λT. (⋃T){is compact in}T)
by auto
  }
  then have ∀A∈Pow(⋃(CoCountable csucc(nat))). ((⋃((CoCountable csucc(nat))
{restricted to} A)) {is compact in} ((CoCountable csucc(nat)) {restricted
to} A))
    ⟶ (A{is in the spectrum of}(λT. (⋃T){is compact in}T)) by auto
  then show thesis unfolding IsAntiComp_def antiProperty_def by auto
qed
```

In conclusion, the cocountable topology defined on `csucc(nat)` is KC but not T_2. Also note that is KC but not anti-hyperconnected, hence KC or US spaces need not to be sober.

The cofinite topology on the natural numbers is T_1, but not US.

```
theorem cofinite_not_US:
  shows ¬((CoFinite nat){is US})
proof
  assume A:(CoFinite nat){is US}
  let N=id(nat)
  have f:N:nat→nat using id_type by auto
  then have fun:N:nat→⋃(CoCardinal(nat,nat)) using union_cocardinal
unfolding Cofinite_def by auto
  then have dom:domain(N)=nat using func1_1_L1 by auto
  with fun have NET:⟨N,Le⟩{is a net on}⋃(CoCardinal(nat,nat)) unfold-
ing IsNet_def
    using Le_directs_nat by auto
  have tot:⋃(CoCardinal(nat,nat))=nat using union_cocardinal by auto
  {
    fix U n assume U:U∈Pow(⋃(CoFinite nat)) n∈Interior(U,(CoFinite nat))
    have Interior(U,(CoFinite nat))∈(CoFinite nat) using topology0.Top_2_L2
      topology0_CoCardinal[OF InfCard_nat] unfolding Cofinite_def by auto
    then have nat-Interior(U,(CoFinite nat))≺nat using U(2) unfolding
Cofinite_def
      CoCardinal_def by auto
    then have Finite(nat-Interior(U,(CoFinite nat))) using lesspoll_nat_is_Finite
by auto moreover
    have nat-U⊆nat-Interior(U,(CoFinite nat)) using topology0.Top_2_L1
      topology0_CoCardinal[OF InfCard_nat] unfolding Cofinite_def by auto
    ultimately have fin:Finite(nat-U) using subset_Finite by auto
    moreover have lin:IsLinOrder(nat,Le) using Le_directs_nat(1) by auto
    then have IsLinOrder(nat-U,Le) using ord_linear_subset[of nat Le
nat-U] by auto
    ultimately have r:nat-U=0 ∨ (∀r∈nat-U. ⟨r,Maximum(Le,nat-U)⟩∈Le)
using linord_max_props(3)[of nat-ULenat-U]
      unfolding FinPow_def by auto
    {
      assume reg:∀s∈nat. ∃r∈nat. ⟨s,r⟩∈Le ∧ Nr∉U
      with r have s:(∀r∈nat-U. ⟨r,Maximum(Le,nat-U)⟩∈Le) nat-U≠0 us-
ing apply_type[OF f] by auto
      have Maximum(Le,nat-U)∈nat using linord_max_props(2)[OF lin _ s(2)]
fin
        unfolding FinPow_def by auto
      then have succ(Maximum(Le,nat-U))∈nat using nat_succI by auto
      with reg have ∃r∈nat. ⟨succ(Maximum(Le,nat-U)),r⟩∈Le ∧ Nr∉U by
auto
      then obtain r where r_def:r∈nat ⟨succ(Maximum(Le,nat-U)),r⟩∈Le
Nr∉U by auto
      from r_def(1,3) have Nr∈nat-U using apply_type[OF f] by auto
      with s(1) have ⟨Nr,Maximum(Le,nat-U)⟩∈Le by auto
      then have ⟨r,Maximum(Le,nat-U)⟩∈Le using id_conv r_def(1) by auto
      then have r<succ(Maximum(Le,nat-U)) by auto
      with r_def(2) have r<r using lt_trans2 by auto
      then have False by auto
```

```
    }
    then have ∃s∈nat. ∀r∈nat. ⟨s,r⟩∈Le ⟶ Nr∈U by auto
  }
  then have ∀n∈nat. ∀U∈Pow(⋃(CoFinite nat)). n∈Interior(U,CoFinite
nat) ⟶ (∃s∈nat. ∀r∈nat. ⟨s,r⟩∈Le ⟶ Nr∈U) by auto
  with tot have ∀n∈⋃(CoCardinal(nat,nat)). ∀U∈Pow(⋃(CoCardinal(nat,nat))).
n∈Interior(U,CoCardinal(nat,nat)) ⟶ (∃s∈nat. ∀r∈nat. ⟨s,r⟩∈Le ⟶ Nr∈U)
    unfolding Cofinite_def by auto
  then have ∀n∈⋃(CoCardinal(nat,nat)). (⟨N,Le⟩→N n {in}(CoCardinal(nat,nat)))
unfolding topology0.NetConverges_def[OF topology0_CoCardinal[OF InfCard_nat]
NET]
    using dom by auto
  with tot have ∀n∈nat. (⟨N,Le⟩→N n {in}(CoFinite nat)) unfolding Cofinite_def
by auto
  then have (⟨N,Le⟩→N 0 {in}(CoFinite nat)) ∧ (⟨N,Le⟩→N 1 {in}(CoFinite
nat)) ∧ 0≠1 by auto
  then show False using A unfolding IsUS_def using fun unfolding Cofinite_def
by auto
qed
```

To end, we need a space which is US but no KC. This example comes from the one point compactification of a T_2, anti-compact and non discrete space. This T_2, anti-compact and non discrete space comes from a construction over the cardinal $\mathbb{N}^+$ or csucc(nat).

```
theorem extension_pow_top:
  shows (Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){is
a topology}
proof-
  have noE:csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  {
    fix M assume M:M⊆(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
    let MP={U∈M. U∈Pow(csucc(nat))}
    let MN={U∈M. U∉Pow(csucc(nat))}
    have unM:⋃M=(⋃MP)∪(⋃MN) by auto
    have csucc(nat)∉csucc(nat) using mem_not_refl by auto
    with M have MN:MN={U∈M. U∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}}
by auto
    have unMP:⋃MP∈Pow(csucc(nat)) by auto
    then have MN=0⟶⋃M∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
      using unM by auto moreover
    {
      assume MN≠0
      with MN have {U∈M. U∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}}≠0
by auto
      then obtain U where U:U∈M U∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}
by blast
      then obtain S where S:U={csucc(nat)}∪S S∈(CoCountable csucc(nat))-{0}
```

```
by auto
      with U MN have csucc(nat)∈U U∈MN by auto
      then have a1:csucc(nat)∈⋃MN by auto
      let SC={S∈(CoCountable csucc(nat)). {csucc(nat)}∪S∈M}
      have unSC:⋃SC∈(CoCountable csucc(nat)) using CoCar_is_topology[OF
InfCard_csucc[OF InfCard_nat]]
        unfolding IsATopology_def unfolding Cocountable_def by blast
      {
        fix s assume s∈{csucc(nat)}∪⋃SC
        then have s=csucc(nat)∨s∈⋃SC by auto
        then have s∈⋃MN∨(∃S∈SC. s∈S) using a1 by auto
        then have s∈⋃MN∨(∃S∈(CoCountable csucc(nat)). {csucc(nat)}∪S∈M
∧ s∈S) by auto
        with MN have s∈⋃MN∨(∃S∈(CoCountable csucc(nat)). {csucc(nat)}∪S∈MN
∧ s∈S) by auto
        then have s∈⋃MN by blast
      }
      then have {csucc(nat)}∪⋃SC⊆⋃MN by blast
      moreover
      {
        fix s assume s∈⋃MN
        then obtain U where U:s∈U U∈M U∉Pow(csucc(nat)) by auto
        with M have U∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))} by
auto
        then obtain S where S:U={csucc(nat)}∪S S∈(CoCountable csucc(nat))
by auto
        with U(1) have s=csucc(nat)∨s∈S by auto
        with S U(2) have s=csucc(nat)∨s∈⋃SC by auto
        then have s∈{csucc(nat)}∪⋃SC by auto
      }
      then have ⋃MN⊆{csucc(nat)}∪⋃SC by blast
      ultimately have unMN:⋃MN={csucc(nat)}∪⋃SC by auto
      from unSC have b1:csucc(nat)-⋃SC≺csucc(nat)∨⋃SC=0 unfolding Cocountable_def
CoCardinal_def
        by auto
      {
        assume 0∈SC
        then have {csucc(nat)}∈M by auto
        then have {csucc(nat)}∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}
using mem_not_refl
          M by auto
        then obtain S where S:S∈(CoCountable csucc(nat))-{0} {csucc(nat)}={csucc(nat)}∪S
by auto
        {
          fix x assume x∈S
          then have x∈{csucc(nat)}∪S by auto
          with S(2) have x∈{csucc(nat)} by auto
          then have x=csucc(nat) by auto
        }
```

```
        then have S⊆{csucc(nat)} by auto
        with S(1) have S={csucc(nat)} by auto
        with S(1) have csucc(nat)-{csucc(nat)}≺csucc(nat) unfolding Cocountable_def
CoCardinal_def
          by auto moreover
        then have csucc(nat)-{csucc(nat)}=csucc(nat) using mem_not_refl[of
csucc(nat)] by force
        ultimately have False by auto
      }
      then have 0∉SC by auto moreover
      from S U(1) have S∈SC by auto
      ultimately have S⊆⋃SC S≠0 by auto
      then have noe:⋃SC≠0 by auto
      with b1 have csucc(nat)-⋃SC≺csucc(nat) by auto
      moreover have csucc(nat)-(⋃SC ∪ ⋃MP)⊆csucc(nat)-⋃SC by auto
      then have csucc(nat)-(⋃SC ∪ ⋃MP)≲csucc(nat)-⋃SC using subset_imp_lepoll
by auto
      ultimately have csucc(nat)-(⋃SC ∪ ⋃MP)≺csucc(nat) using lesspoll_trans1
by auto moreover
      have ⋃SC⊆⋃(CoCountable csucc(nat)) using unSC by auto
      then have ⋃SC⊆csucc(nat) using union_cocardinal[OF noE] unfold-
ing Cocountable_def by auto
      ultimately have (⋃SC ∪ ⋃MP)∈(CoCountable csucc(nat))
        using unMP unfolding Cocountable_def CoCardinal_def by auto
      then have {csucc(nat)}∪(⋃SC ∪ ⋃MP)∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}})
        using noe by auto moreover
      from unM unMN have ⋃M=({csucc(nat)}∪⋃SC) ∪ ⋃MP by auto
      then have ⋃M={csucc(nat)}∪(⋃SC ∪ ⋃MP) by auto
      ultimately have ⋃M∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
    }
    ultimately have ⋃M∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
  }
  then have ∀M∈Pow(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}). ⋃M∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
  moreover
  {
    fix U V assume UV:U∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
    {
      assume csucc(nat)∉U∨csucc(nat)∉V
      with UV have U∈Pow(csucc(nat))∨V∈Pow(csucc(nat)) by auto
      then have U∩V∈Pow(csucc(nat)) by auto
      then have U∩V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
```

```
    }
    moreover
    {
      assume csucc(nat)∈U∧csucc(nat)∈V
      then obtain SU SV where S:U={csucc(nat)}∪SU V={csucc(nat)}∪SV
SU∈(CoCountable csucc(nat))-{0}
        SV∈(CoCountable csucc(nat))-{0} using UV mem_not_refl by auto
      from S(1,2) have U∩V={csucc(nat)}∪(SU∩SV) by auto moreover
      from S(3,4) have SU∩SV∈(CoCountable csucc(nat)) using CoCar_is_topology[OF
InfCard_csucc[OF InfCard_nat]] unfolding IsATopology_def
        unfolding Cocountable_def by blast moreover
      from S(3,4) have SU∩SV≠0 using cocountable_in_csucc_nat_HConn un-
folding IsHConnected_def
        by auto ultimately
      have U∩V∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}} by auto
      then have U∩V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
    }
    ultimately have U∩V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
  }
  then have ∀U∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}).
∀V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}).
U∩V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
by auto
  ultimately show thesis unfolding IsATopology_def by auto
qed
```

This topology is defined over $\mathbb{N}^+ \cup \{\mathbb{N}^+\}$ or csucc(nat)∪{csucc(nat)}.

```
lemma extension_pow_union:
  shows ⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})=csucc(nat)∪
proof
  have noE:csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  have ⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})=⋃(Pow(csucc
∪ (⋃{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
    by blast
  also have ...=csucc(nat) ∪ (⋃{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
by auto
  ultimately have A:⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})=csucc(nat) ∪ (⋃{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
by auto
  have ⋃(CoCountable csucc(nat))∈(CoCountable csucc(nat)) using CoCar_is_topology[OF
InfCard_csucc[OF InfCard_nat]]
    unfolding IsATopology_def Cocountable_def by auto
  then have csucc(nat)∈(CoCountable csucc(nat)) using union_cocardinal[OF
noE] unfolding Cocountable_def
    by auto
  with noE have csucc(nat)∈(CoCountable csucc(nat))-{0} by auto
  then have {csucc(nat)}∪csucc(nat)∈{{csucc(nat)}∪S. S∈(CoCountable
```

```
csucc(nat))-{0}} by auto
  then have {csucc(nat)}∪csucc(nat)⊆⋃{{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} by blast
  with A show csucc(nat)∪{csucc(nat)}⊆⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}})
    by auto
  {
    fix x assume x:x∈(⋃{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
x≠csucc(nat)
    then obtain U where U:U∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}
x∈U by blast
    then obtain S where S:U={csucc(nat)}∪S S∈(CoCountable csucc(nat))-{0}
by auto
    with U(2) x(2) have x∈S by auto
    with S(2) have x∈⋃(CoCountable csucc(nat)) by auto
    then have x∈csucc(nat) using union_cocardinal[OF noE] unfolding Cocountable_def
by auto
  }
  then have (⋃{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})⊆csucc(nat)
∪{csucc(nat)} by blast
  with A show ⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})⊆csuc
    by blast
qed
```

This topology has a discrete open subspace.

```
lemma extension_pow_subspace:
  shows (Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricted
to}csucc(nat)=Pow(csucc(nat))
  and csucc(nat)∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
proof
  show csucc(nat)∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) by auto
  {
    fix x assume x∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}csucc(nat)
    then obtain R where x=csucc(nat)∩R R∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}}) unfolding RestrictedTo_def
      by auto
    then have x∈Pow(csucc(nat)) by auto
  }
  then show (Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricte
to}csucc(nat)⊆Pow(csucc(nat)) by auto
  {
    fix x assume x:x∈Pow(csucc(nat))
    then have x=csucc(nat)∩x by auto
    with x have x∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}csucc(nat)
      unfolding RestrictedTo_def by auto
```

```
  }
  then show Pow(csucc(nat))⊆(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}csucc(nat) by auto
qed
```

This topology is Hausdorff.

```
theorem extension_pow_T2:
  shows (Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){is
T2}
proof-
  have noE:csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  {
    fix A B assume A∈⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
      B∈⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
A≠B
    then have AB:A∈csucc(nat)∪{csucc(nat)} B∈csucc(nat)∪{csucc(nat)}
A≠B using extension_pow_union by auto
    {
      assume A≠csucc(nat) B≠csucc(nat)
      then have A∈csucc(nat) B∈csucc(nat) using AB by auto
      then have sub:{A}∈Pow(csucc(nat)) {B}∈Pow(csucc(nat)) by auto
      then have {A}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}csucc(nat)
        {B}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricted
to}csucc(nat) using extension_pow_subspace(1)
        by auto
      then obtain RA RB where {A}=csucc(nat)∩RA {B}=csucc(nat)∩RB RA∈(Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
        RB∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
unfolding RestrictedTo_def by auto
      then have {A}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}) {B}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
        using extension_pow_subspace(2) extension_pow_top unfolding IsATopology_def
by auto
      moreover
      from AB(3) have {A}∩{B}=0 by auto ultimately
      have ∃U∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}).
∃V∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}).
A∈U∧B∈V∧U∩V=0 by auto
    }
    moreover
    {
      assume A=csucc(nat)∨B=csucc(nat)
      with AB(3) have disj:(A=csucc(nat)∧B≠csucc(nat))∨(B=csucc(nat)∧A≠csucc(nat))
by auto
      {
        assume ass:A=csucc(nat)∧B≠csucc(nat)
```

```
        then have p:B∈csucc(nat) using AB(2) by auto
        have {B}≈1 using singleton_eqpoll_1 by auto
        then have {B}≺nat using eq_lesspoll_trans n_lesspoll_nat by auto
        then have {B}≲nat using lesspoll_imp_lepoll by auto
        then have {B}≺csucc(nat) using Card_less_csucc_eq_le[OF Card_nat]
by auto
        with p have {B}{is closed in}(CoCountable csucc(nat)) unfold-
ing Cocountable_def
          using closed_sets_cocardinal[OF noE] by auto
        then have csucc(nat)-{B}∈(CoCountable csucc(nat)) unfolding IsClosed_def
          Cocountable_def using union_cocardinal[OF noE] by auto more-
over
        {
          assume csucc(nat)-{B}=0
          with p have csucc(nat)={B} by auto
          then have csucc(nat)≈1 using singleton_eqpoll_1 by auto
          then have csucc(nat)≲nat using eq_lesspoll_trans n_lesspoll_nat
lesspoll_imp_lepoll by auto
          then have csucc(nat)≺csucc(nat) using Card_less_csucc_eq_le[OF
Card_nat] by auto
          then have False by auto
        }
        ultimately
        have {csucc(nat)}∪(csucc(nat)-{B})∈{{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} by auto
        then have U1:{csucc(nat)}∪(csucc(nat)-{B})∈(Pow(csucc(nat)) ∪
{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) by auto
        have {B}∈Pow(csucc(nat)) using p by auto
        then have {B}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}csucc(nat)
          using extension_pow_subspace(1) by auto
        then obtain R where R∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} {B}=csucc(nat)∩R
          unfolding RestrictedTo_def by auto
        then have U2:{B}∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} using extension_pow_subspace(2)
          extension_pow_top unfolding IsATopology_def by auto
        have ({csucc(nat)}∪(csucc(nat)-{B}))∩{B}=0 using p mem_not_refl[of
csucc(nat)] by auto
        with U1 U2 have ∃U∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}. ∃V∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}.
          A∈U∧B∈V∧U∩V=0 using ass(1) by auto
      }
      moreover
      {
        assume ¬(A=csucc(nat)∧B≠csucc(nat))
        then have ass:B = csucc(nat) ∧ A ≠ csucc(nat) using disj by
auto
```

```
        then have p:A∈csucc(nat) using AB(1) by auto
        have {A}≈1 using singleton_eqpoll_1 by auto
        then have {A}≺nat using eq_lesspoll_trans n_lesspoll_nat by auto
        then have {A}≲nat using lesspoll_imp_lepoll by auto
        then have {A}≺csucc(nat) using Card_less_csucc_eq_le[OF Card_nat]
by auto
        with p have {A}{is closed in}(CoCountable csucc(nat)) unfold-
ing Cocountable_def
          using closed_sets_cocardinal[OF noE] by auto
        then have csucc(nat)-{A}∈(CoCountable csucc(nat)) unfolding IsClosed_def
          Cocountable_def using union_cocardinal[OF noE] by auto more-
over
        {
          assume csucc(nat)-{A}=0
          with p have csucc(nat)={A} by auto
          then have csucc(nat)≈1 using singleton_eqpoll_1 by auto
          then have csucc(nat)≲nat using eq_lesspoll_trans n_lesspoll_nat
lesspoll_imp_lepoll by auto
          then have csucc(nat)≺csucc(nat) using Card_less_csucc_eq_le[OF
Card_nat] by auto
          then have False by auto
        }
        ultimately
        have {csucc(nat)}∪(csucc(nat)-{A})∈{{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} by auto
        then have U1:{csucc(nat)}∪(csucc(nat)-{A})∈(Pow(csucc(nat)) ∪
{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) by auto
        have {A}∈Pow(csucc(nat)) using p by auto
        then have {A}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}csucc(nat)
          using extension_pow_subspace(1) by auto
        then obtain R where R∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} {A}=csucc(nat)∩R
          unfolding RestrictedTo_def by auto
        then have U2:{A}∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}} using extension_pow_subspace(2)
          extension_pow_top unfolding IsATopology_def by auto
        have int:{A}∩({csucc(nat)}∪(csucc(nat)-{A}))=0 using p mem_not_refl[of
csucc(nat)] by auto
        have A∈{A} csucc(nat)∈({csucc(nat)}∪(csucc(nat)-{A})) by auto
        with int U1 have ∃V∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}.
          A∈{A}∧csucc(nat)∈V∧{A}∩V=0 by auto
        with U2 have ∃U∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}. ∃V∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}.
          A∈U∧csucc(nat)∈V∧U∩V=0 using exI[where P=λU. U∈Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}∧(∃V∈Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}.
```

```
          A∈U∧csucc(nat)∈V∧U∩V=0) and x={A}] unfolding Bex_def by auto
        then have ∃U∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}. ∃V∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}.
          A∈U∧B∈V∧U∩V=0 using ass by auto
      }
      ultimately have ∃U∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}. ∃V∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}.
          A∈U∧B∈V∧U∩V=0 by auto
    }
    ultimately have ∃U∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}. ∃V∈Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}.
          A∈U∧B∈V∧U∩V=0 by auto
  }
  then show thesis unfolding isT2_def by auto
qed
```

The topology we built is not discrete; i.e., not every set is open.

```
theorem extension_pow_notDiscrete:
  shows {csucc(nat)}∉(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
proof
  assume {csucc(nat)}∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
  then have {csucc(nat)}∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}
using mem_not_refl by auto
  then obtain S where S:S∈(CoCountable csucc(nat))-{0} {csucc(nat)}={csucc(nat)}∪S
by auto
  {
    fix x assume x∈S
    then have x∈{csucc(nat)}∪S by auto
    with S(2) have x∈{csucc(nat)} by auto
    then have x=csucc(nat) by auto
  }
  then have S⊆{csucc(nat)} by auto
  with S(1) have S={csucc(nat)} by auto
  with S(1) have csucc(nat)-{csucc(nat)}≺csucc(nat) unfolding Cocountable_def
CoCardinal_def
    by auto moreover
  then have csucc(nat)-{csucc(nat)}=csucc(nat) using mem_not_refl[of
csucc(nat)] by force
  ultimately show False by auto
qed
```

The topology we built is anti-compact.

```
theorem extension_pow_antiCompact:
  shows (Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){is
```

```
anti-compact}
proof-
  have noE:csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  {
    fix K assume K:K⊆⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}})
      (⋃((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricted
to}K)){is compact in}((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}K)
    from K(1) have sub:K⊆csucc(nat) ∪{csucc(nat)} using extension_pow_union
by auto
    have (⋃((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricte
to}K))=(csucc(nat) ∪{csucc(nat)})∩K
      using extension_pow_union unfolding RestrictedTo_def by auto more-
over
    from sub have (csucc(nat) ∪{csucc(nat)})∩K=K by auto
    ultimately have (⋃((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}){restricted to}K))=K by auto
    with K(2) have K{is compact in}((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}}){restricted to}K) by auto
    then have comp:K{is compact in}(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}}) using
      compact_subspace_imp_compact by auto
    {
      assume ss:K⊆csucc(nat)
      then have K{is compact in}((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}}){restricted to}csucc(nat))
        using compact_imp_compact_subspace comp Compact_is_card_nat by
auto
      then have K{is compact in}Pow(csucc(nat)) using extension_pow_subspace(1)
by auto
      then have K{is compact in}(Pow(csucc(nat)){restricted to}K) us-
ing compact_imp_compact_subspace
        Compact_is_card_nat by auto moreover
      have ⋃(Pow(csucc(nat)){restricted to}K)=K using ss unfolding RestrictedTo_def
by auto
      ultimately have (⋃(Pow(csucc(nat)){restricted to}K)){is compact
in}(Pow(csucc(nat)){restricted to}K) by auto
      then have K{is in the spectrum of}(λT. (⋃T){is compact in}T) us-
ing pow_anti_compact
        unfolding IsAntiComp_def antiProperty_def using ss by auto
    }
    moreover
    {
      assume ¬(K⊆csucc(nat))
      with sub have csucc(nat)∈K by auto
      with sub have ss:K-{csucc(nat)}⊆csucc(nat) by auto
      {
        assume prec:K-{csucc(nat)}≺csucc(nat)
```

```
        then have (K-{csucc(nat)}){is closed in}(CoCountable csucc(nat))
          using closed_sets_cocardinal[OF noE] ss unfolding Cocountable_def
by auto
        then have csucc(nat)-(K-{csucc(nat)})∈(CoCountable csucc(nat))
unfolding IsClosed_def
          Cocountable_def using union_cocardinal[OF noE] by auto more-
over
        {
          assume csucc(nat)-(K-{csucc(nat)})=0
          with ss have csucc(nat)=(K-{csucc(nat)}) by auto
          with prec have False by auto
        }
        ultimately have {csucc(nat)} ∪(csucc(nat)-(K-{csucc(nat)}))∈{{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}}
          by auto
        moreover have {csucc(nat)} ∪(csucc(nat)-(K-{csucc(nat)}))=({csucc(nat)}
∪ csucc(nat))-(K-{csucc(nat)}) by blast
        ultimately have ({csucc(nat)} ∪ csucc(nat))-(K-{csucc(nat)})∈{{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}} by auto
        then have ({csucc(nat)} ∪ csucc(nat))-(K-{csucc(nat)})∈(Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
          by auto moreover
        have csucc(nat) ∪ {csucc(nat)}={csucc(nat)} ∪ csucc(nat) by auto
        ultimately have (csucc(nat) ∪ {csucc(nat)})-(K-{csucc(nat)})∈(Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
          by auto
        then have (⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable
csucc(nat))-{0}}))-(K-{csucc(nat)})∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}})
          using extension_pow_union by auto
        then have (K-{csucc(nat)}){is closed in}(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}})
          unfolding IsClosed_def using ss by auto
        with comp have (K∩(K-{csucc(nat)})){is compact in}(Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) using compact_closed
          Compact_is_card_nat by auto
        moreover have K∩(K-{csucc(nat)})=(K-{csucc(nat)}) by auto
        ultimately have (K-{csucc(nat)}){is compact in}(Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) by auto
        with ss have (K-{csucc(nat)}){is compact in}((Pow(csucc(nat))
∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricted to}csucc(nat))
          using compact_imp_compact_subspace comp Compact_is_card_nat
by auto
        then have (K-{csucc(nat)}){is compact in}(Pow(csucc(nat))) us-
ing extension_pow_subspace(1) by auto
        then have (K-{csucc(nat)}){is compact in}(Pow(csucc(nat)){restricted
to}(K-{csucc(nat)})) using compact_imp_compact_subspace
          Compact_is_card_nat by auto moreover
        have ⋃(Pow(csucc(nat)){restricted to}(K-{csucc(nat)}))=(K-{csucc(nat)})
```

```
using ss unfolding RestrictedTo_def by auto
        ultimately have (⋃(Pow(csucc(nat)){restricted to}(K-{csucc(nat)}))){is
compact in}(Pow(csucc(nat)){restricted to}(K-{csucc(nat)})) by auto
        then have (K-{csucc(nat)}){is in the spectrum of}(λT. (⋃T){is
compact in}T) using pow_anti_compact
          unfolding IsAntiComp_def antiProperty_def using ss by auto
        then have Finite(K-{csucc(nat)}) using compact_spectrum by auto
moreover
        have Finite({csucc(nat)}) by auto ultimately
        have Finite(K) using Diff_Finite[of {csucc(nat)} K] by auto
        then have K{is in the spectrum of}(λT. (⋃T){is compact in}T)
using compact_spectrum by auto
      }
      moreover
      {
        assume ¬(K-{csucc(nat)}≺csucc(nat))
        with ss have K-{csucc(nat)}≈csucc(nat) using lepoll_iff_leqpoll
subset_imp_lepoll[of K-{csucc(nat)}
          csucc(nat)] by auto
        then have csucc(nat)≈K-{csucc(nat)} using eqpoll_sym by auto
        then have nat≺K-{csucc(nat)} using lesspoll_eq_trans lt_csucc[OF
Ord_nat]
          lt_Card_imp_lesspoll[OF Card_csucc[OF Ord_nat]] by auto
        then have nat≲K-{csucc(nat)} using lepoll_iff_leqpoll by auto
        then obtain f where f∈inj(nat,K-{csucc(nat)}) unfolding lepoll_def
by auto moreover
        then have fun:f:nat→K-{csucc(nat)} unfolding inj_def by auto
        then have f∈surj(nat,range(f)) using fun_is_surj by auto
        ultimately have f∈bij(nat,range(f)) unfolding bij_def inj_def
surj_def by auto
        then have nat≈range(f) unfolding eqpoll_def by auto
        then have e:range(f)≈nat using eqpoll_sym by auto
        then have as2:range(f)≺csucc(nat) using lt_Card_imp_lesspoll[OF
Card_csucc[OF Ord_nat]]
          lt_csucc[OF Ord_nat] eq_lesspoll_trans by auto
        then have range(f){is closed in}(CoCountable csucc(nat)) using
closed_sets_cocardinal[of csucc(nat)
            range(f)csucc(nat)] unfolding Cocountable_def using func1_1_L5B[OF
fun] ss noE by auto
        then have csucc(nat)-(range(f))∈(CoCountable csucc(nat)) un-
folding IsClosed_def
          Cocountable_def using union_cocardinal[OF noE] by auto more-
over
        {
          assume csucc(nat)-(range(f))=0
          with ss func1_1_L5B[OF fun] have csucc(nat)=(range(f)) by blast
          with as2 have False by auto
        }
        ultimately have {csucc(nat)} ∪(csucc(nat)-(range(f)))∈{{csucc(nat)}∪S.
```

S∈(CoCountable csucc(nat))-{0}}
 by auto
 moreover have {csucc(nat)} ∪(csucc(nat)-(range(f)))=({csucc(nat)} ∪ csucc(nat))-(range(f)) **using** func1_1_L5B[OF fun] **by** blast
 ultimately have ({csucc(nat)} ∪ csucc(nat))-(range(f))∈{{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}} **by** auto
 then have ({csucc(nat)} ∪ csucc(nat))-(range(f))∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
 by auto **moreover**
 have csucc(nat) ∪ {csucc(nat)}={csucc(nat)} ∪ csucc(nat) **by** auto
 ultimately have (csucc(nat) ∪ {csucc(nat)})-(range(f))∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
 by auto
 then have (⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}))-(range(f))∈(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
 using extension_pow_union **by** auto **moreover**
 have range(f)⊆⋃(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) **using** ss func1_1_L5B[OF fun] **by** auto
 ultimately have (range(f)){is closed in}(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
 unfolding IsClosed_def **by** blast
 with comp **have** (K∩(range(f))){is compact in}(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) **using** compact_closed Compact_is_card_nat **by** auto
 moreover have K∩(range(f))=(range(f)) **using** func1_1_L5B[OF fun] **by** auto
 ultimately have (range(f)){is compact in}(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}) **by** auto
 with ss func1_1_L5B[OF fun] **have** (range(f)){is compact in}((Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){restricted to}csucc(nat))
 using compact_imp_compact_subspace[of range(f) nat Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}} csucc(nat)] comp Compact_is_card_nat **by** auto
 then have (range(f)){is compact in}(Pow(csucc(nat))) **using** extension_pow_subspace(1 **by** auto
 then have (range(f)){is compact in}(Pow(csucc(nat)){restricted to}(range(f))) **using** compact_imp_compact_subspace Compact_is_card_nat **by** auto **moreover**
 have ⋃(Pow(csucc(nat)){restricted to}(range(f)))=(range(f)) **using** ss func1_1_L5B[OF fun] **unfolding** RestrictedTo_def **by** auto
 ultimately have (⋃(Pow(csucc(nat)){restricted to}(range(f)))){is compact in}(Pow(csucc(nat)){restricted to}(range(f))) **by** auto
 then have (range(f)){is in the spectrum of}(λT. (⋃T){is compact in}T) **using** pow_anti_compact[of csucc(nat)]
 unfolding IsAntiComp_def antiProperty_def **using** ss func1_1_L5B[OF fun] **by** auto
 then have Finite(range(f)) **using** compact_spectrum **by** auto **moreover**

```
        then have Finite(nat) using e eqpoll_imp_Finite_iff by auto ul-
timately
        have False using nat_not_Finite by auto
      }
      ultimately have K {is in the spectrum of}( λT. (⋃T) {is compact
in} T) by auto
    }
    ultimately have K {is in the spectrum of}( λT. (⋃T) {is compact in}
T) by auto
  }
  then show thesis unfolding IsAntiComp_def antiProperty_def by auto
qed
```

If a topological space is KC, then its one-point compactification is US.

```
theorem (in topology0) KC_imp_OP_comp_is_US:
  assumes T{is KC}
  shows ({one-point compactification of}T){is US}
proof-
  {
    fix N x y assume A:N:nat→⋃({one-point compactification of}T) ⟨N,Le⟩→N
x{in}({one-point compactification of}T) ⟨N,Le⟩→N y{in}({one-point compactification
of}T) x≠y
    have dir:Le directs nat using Le_directs_nat(2).
    from A(1) have dom:domain(N)=nat using func1_1_L1 by auto
    with dir A(1) have NET:⟨N,Le⟩{is a net on}⋃({one-point compactification
of}T) unfolding IsNet_def by auto
    have xy:x∈⋃({one-point compactification of}T) y∈⋃({one-point compactification
of}T)
      using A(2,3) topology0.NetConverges_def[OF _ NET] unfolding topology0_def
using op_comp_is_top dom by auto
    then have pp:x∈⋃T ∪{⋃T} y∈⋃T ∪{⋃T} using op_compact_total by
auto
    from A(2) have comp:∀U∈Pow(⋃{one-point compactification of}T).
        x ∈ Interior(U, {one-point compactification of}T) ⟶
        (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ U) using topology0.NetConverges_def[OF
_ NET, of x]
        unfolding topology0_def using op_comp_is_top dom op_compact_total
by auto
    from A(3) have op2:∀U∈Pow(⋃{one-point compactification of}T).
        y ∈ Interior(U, {one-point compactification of}T) ⟶
        (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ U) using topology0.NetConverges_def[OF
_ NET, of y]
        unfolding topology0_def using op_comp_is_top dom op_compact_total
by auto
    {
      assume p:x∈⋃T y∈⋃T
      {
        assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
        have ⋃T∈({one-point compactification of}T) using open_subspace
```

```
by auto
        then have ⋃T=Interior(⋃T,{one-point compactification of}T) us-
ing topology0.Top_2_L3
          unfolding topology0_def using op_comp_is_top by auto
        then have x∈Interior(⋃T,{one-point compactification of}T) us-
ing p(1) by auto moreover
        have ⋃T∈Pow(⋃({one-point compactification of}T)) using open_subspace(1)
by auto
        ultimately have ∃t∈domain(fst(⟨N, Le⟩)). ∀m∈domain(fst(⟨N, Le⟩)).
⟨t, m⟩ ∈ snd(⟨N, Le⟩) ⟶ fst(⟨N, Le⟩)  m ∈ ⋃T using A(2)
          using topology0.NetConverges_def[OF _ NET] op_comp_is_top un-
folding topology0_def by blast
        then have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ ⋃T using dom
by auto
        then obtain t where t:t∈nat ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ ⋃T
by auto
        from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T by
auto
        from t(1) n(1) dir obtain z where z:z∈nat ⟨n,z⟩∈Le ⟨t,z⟩∈Le un-
folding IsDirectedSet_def
          by auto
        from t(2) z(1,3) have Nz∈⋃T by auto moreover
        from n(2) z(1,2) have Nz=⋃T by auto ultimately
        have False using mem_not_refl by auto
      }
      then have reg:∀n∈nat. ∃m∈nat. Nm≠⋃T ∧ ⟨n,m⟩∈Le by auto
      let NN={⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩. n∈nat}
      {
        fix x z assume A1:⟨x, z⟩ ∈ NN
        {
          fix y' assume A2:⟨x,y'⟩∈NN
          with A1 have z=y' by auto
        }
        then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
      }
      then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
      moreover
      {
      fix n assume as:n∈nat
        with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
        then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧
⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
          nat_into_Ord[of m] by auto
        then have (μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈nat by auto
        then have N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈⋃({one-point compactification
of}T) using apply_type[OF A(1)] op_compact_total by auto
        with as have ⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃({one-point
compactification of}T) by auto
      }
```

```
      then have NN∈Pow(nat×⋃({one-point compactification of}T)) by
auto
      ultimately have NFun:NN:nat→⋃({one-point compactification of}T)
unfolding Pi_def function_def domain_def by auto
      {
        fix n assume as:n∈nat
        with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
        then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧
⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
          nat_into_Ord[of m] by auto
        then have NNn≠⋃T using apply_equality[OF _ NFun] by auto
      }
      then have noy:∀n∈nat. NNn≠⋃T by auto
      then have ∀n∈nat. NNn∈⋃T using apply_type[OF NFun] op_compact_total
by auto
      then have R:NN:nat→⋃T using func1_1_L1A[OF NFun] by auto
      have dom2:domain(NN)=nat by auto
      then have net2:⟨NN,Le⟩{is a net on}⋃T unfolding IsNet_def using
R dir by auto
      {
        fix U assume U:U⊆⋃T x∈int(U)
        have intT:int(U)∈T using Top_2_L2 by auto
        then have int(U)∈({one-point compactification of}T) unfolding
OPCompactification_def
          by auto
        then have Interior(int(U),{one-point compactification of}T)=int(U)
using topology0.Top_2_L3
          unfolding topology0_def using op_comp_is_top by auto
        with U(2) have x∈Interior(int(U),{one-point compactification
of}T) by auto
        with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) us-
ing comp op_compact_total by auto
        then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U us-
ing Top_2_L1 by auto
        {
          fix s assume AA:⟨r,s⟩∈Le
          with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
          then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm.
Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
            nat_into_Ord by auto
          with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by
auto
          with r_def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
          then have NNs∈U using apply_equality[OF _ NFun] AA by auto
        }
        then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
        with r_def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
      }
      then have ∀U∈Pow(⋃T). x ∈ int(U)
```

```
        ⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN  s ∈ U) by auto
      then have conx:⟨NN,Le⟩→N x{in}T using NetConverges_def[OF net2]
p(1) op_comp_is_top
        unfolding topology0_def using xy(1) dom2 by auto
      {
        fix U assume U:U⊆⋃T y∈int(U)
        have intT:int(U)∈T using Top_2_L2 by auto
        then have int(U)∈({one-point compactification of}T) unfolding
OPCompactification_def
          by auto
        then have Interior(int(U),{one-point compactification of}T)=int(U)
using topology0.Top_2_L3
          unfolding topology0_def using op_comp_is_top by auto
        with U(2) have y∈Interior(int(U),{one-point compactification
of}T) by auto
        with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) us-
ing op2 op_compact_total by auto
        then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U us-
ing Top_2_L1 by auto
        {
          fix s assume AA:⟨r,s⟩∈Le
          with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
          then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm.
Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
            nat_into_Ord by auto
          with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by
auto
          with r_def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
          then have NNs∈U using apply_equality[OF _ NFun] AA by auto
        }
        then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
        with r_def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
      }
      then have ∀U∈Pow(⋃T). y ∈ int(U)
        ⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN  s ∈ U) by auto
      then have cony:⟨NN,Le⟩→N y{in}T using NetConverges_def[OF net2]
p(2) op_comp_is_top
        unfolding topology0_def using xy(2) dom2 by auto
      with conx assms have x=y using KC_imp_US unfolding IsUS_def us-
ing R by auto
      with A(4) have False by auto
    }
    moreover
    {
      assume AAA:x∉⋃T∨y∉⋃T
      with pp have x=⋃T∨y=⋃T by auto
      {
        assume x:x=⋃T
        with A(4) have y:y∈⋃T using pp(2) by auto
```

```
        {
          assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
          have ⋃T∈({one-point compactification of}T) using open_subspace
by auto
          then have ⋃T=Interior(⋃T,{one-point compactification of}T)
using topology0.Top_2_L3
            unfolding topology0_def using op_comp_is_top by auto
          then have y∈Interior(⋃T,{one-point compactification of}T)
using y by auto moreover
          have ⋃T∈Pow(⋃({one-point compactification of}T)) using open_subspace(1)
by auto
          ultimately have ∃t∈domain(fst(⟨N, Le⟩)). ∀m∈domain(fst(⟨N,
Le⟩)). ⟨t, m⟩ ∈ snd(⟨N, Le⟩) ⟶ fst(⟨N, Le⟩)  m ∈ ⋃T using A(3)
            using topology0.NetConverges_def[OF _ NET] op_comp_is_top
unfolding topology0_def by blast
          then have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ ⋃T using
dom by auto
          then obtain t where t:t∈nat ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈
⋃T by auto
          from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
by auto
          from t(1) n(1) dir obtain z where z:z∈nat ⟨n,z⟩∈Le ⟨t,z⟩∈Le
unfolding IsDirectedSet_def
            by auto
          from t(2) z(1,3) have Nz∈⋃T by auto moreover
          from n(2) z(1,2) have Nz=⋃T by auto ultimately
          have False using mem_not_refl by auto
        }
        then have reg:∀n∈nat. ∃m∈nat. Nm≠⋃T ∧ ⟨n,m⟩∈Le by auto
        let NN={⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩. n∈nat}
        {
          fix x z assume A1:⟨x, z⟩ ∈ NN
          {
            fix y' assume A2:⟨x,y'⟩∈NN
            with A1 have z=y' by auto
          }
          then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
        }
        then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by
auto
        moreover
        {
          fix n assume as:n∈nat
          with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
          then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T
∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
            nat_into_Ord[of m] by auto
          then have (μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈nat by auto
          then have N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈⋃({one-point compactification
```

```
of}T) using apply_type[OF A(1)] op_compact_total by auto
          with as have ⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃({one-point
compactification of}T) by auto
        }
        then have NN∈Pow(nat×⋃({one-point compactification of}T)) by
auto
        ultimately have NFun:NN:nat→⋃({one-point compactification of}T)
unfolding Pi_def function_def domain_def by auto
        {
          fix n assume as:n∈nat
          with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
          then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T
∧ ⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
            nat_into_Ord[of m] by auto
          then have NNn≠⋃T using apply_equality[OF _ NFun] by auto
        }
        then have noy:∀n∈nat. NNn≠⋃T by auto
        then have ∀n∈nat. NNn∈⋃T using apply_type[OF NFun] op_compact_total
by auto
        then have R:NN:nat→⋃T using func1_1_L1A[OF NFun] by auto
        have dom2:domain(NN)=nat by auto
        then have net2:⟨NN,Le⟩{is a net on}⋃T unfolding IsNet_def us-
ing R dir by auto
        {
          fix U assume U:U⊆⋃T y∈int(U)
          have intT:int(U)∈T using Top_2_L2 by auto
          then have int(U)∈({one-point compactification of}T) unfold-
ing OPCompactification_def
            by auto
          then have Interior(int(U),{one-point compactification of}T)=int(U)
using topology0.Top_2_L3
            unfolding topology0_def using op_comp_is_top by auto
          with U(2) have y∈Interior(int(U),{one-point compactification
of}T) by auto
          with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) us-
ing op2 op_compact_total by auto
          then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U
using Top_2_L1 by auto
          {
            fix s assume AA:⟨r,s⟩∈Le
            with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
            then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm.
Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
              nat_into_Ord by auto
            with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans
by auto
            with r_def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
            then have NNs∈U using apply_equality[OF _ NFun] AA by auto
          }
```

```
          then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
          with r_def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
        }
        then have ∀U∈Pow(⋃T). y ∈ int(U)
          ⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN  s ∈ U) by auto
        then have cony:⟨NN,Le⟩→N y{in}T using NetConverges_def[OF net2]
y op_comp_is_top
          unfolding topology0_def using xy(2) dom2 by auto
        let A={y}∪NNnat
        {
          fix M assume Acov:A⊆⋃M M⊆T
          then have y∈⋃M by auto
          then obtain V where V:y∈V V∈M by auto
          with Acov(2) have VT:V∈T by auto
          then have V=int(V) using Top_2_L3 by auto
          with V(1) have y∈int(V) by auto
          with cony obtain r where rr:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈V
            unfolding NetConverges_def[OF net2, of y] using dom2 VT y
by auto
          have NresFun:restrict(NN,{n∈nat. ⟨n,r⟩∈Le}):{n∈nat. ⟨n,r⟩∈Le}→⋃T
using restrict_fun
            [OF R, of {n∈nat. ⟨n,r⟩∈Le}] by auto
          then have restrict(NN,{n∈nat. ⟨n,r⟩∈Le})∈surj({n∈nat. ⟨n,r⟩∈Le},range(restrict(NN,{n∈nat.
⟨n,r⟩∈Le}))
            using fun_is_surj by auto moreover
          have {n∈nat. ⟨n,r⟩∈Le}⊆nat by auto
          then have {n∈nat. ⟨n,r⟩∈Le}≲nat using subset_imp_lepoll by
auto
          ultimately have range(restrict(NN,{n∈nat. ⟨n,r⟩∈Le}))≲{n∈nat.
⟨n,r⟩∈Le} using surj_fun_inv_2 by auto
          moreover
          have {n∈nat. ⟨n,0⟩∈Le}={0} by auto
          then have Finite({n∈nat. ⟨n,0⟩∈Le}) by auto moreover
          {
            fix j assume as:j∈nat Finite({n∈nat. ⟨n,j⟩∈Le})
            {
              fix t assume t∈{n∈nat. ⟨n,succ(j)⟩∈Le}
              then have t∈nat ⟨t,succ(j)⟩∈Le by auto
              then have t≤succ(j) by auto
              then have t⊆succ(j) using le_imp_subset by auto
              then have t⊆j ∪{j} using succ_explained by auto
              then have j∈t∨t⊆j by auto
              then have j∈t∨t≤j using subset_imp_le ‹t∈nat› ‹j∈nat›
nat_into_Ord by auto
              then have j ∪{j}⊆t∨t≤j using ‹t∈nat› ‹j∈nat› nat_into_Ord
unfolding Ord_def
                Transset_def by auto
              then have succ(j)⊆t∨t≤j using succ_explained by auto
              with ‹t⊆succ(j)› have t=succ(j)∨t≤j by auto
```

```
          with ‹t∈nat› ‹j∈nat› have t∈{n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}
by auto
        }
        then have {n∈nat. ⟨n,succ(j)⟩∈Le} ⊆{n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}
by auto
        moreover have Finite({n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}) using
as(2) Finite_cons
          by auto
        ultimately have Finite({n∈nat. ⟨n,succ(j)⟩∈Le}) using subset_Finite
by auto
      }
      then have ∀j∈nat. Finite({n∈nat. ⟨n,j⟩∈Le}) ⟶ Finite({n∈nat.
⟨n,succ(j)⟩∈Le})
        by auto
      ultimately have Finite(range(restrict(NN, {n ∈ nat . ⟨n, r⟩
∈ Le})))
        using lepoll_Finite[of range(restrict(NN, {n ∈ nat . ⟨n, r⟩
∈ Le}))
          {n ∈ nat . ⟨n, r⟩ ∈ Le}] ind_on_nat[OF ‹r∈nat›, where P=λt.
Finite({n∈nat. ⟨n,t⟩∈Le})] by auto
      then have Finite((restrict(NN, {n ∈ nat . ⟨n, r⟩ ∈ Le}))``{n
∈ nat . ⟨n, r⟩ ∈ Le}) using range_image_domain[OF NresFun]
        by auto
      then have Finite(NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}) using restrict_image
by auto
      then have (NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}){is in the spectrum of}(λT.
(⋃T){is compact in}T) using compact_spectrum by auto
      moreover have ⋃(T{restricted to}NN``{n ∈ nat . ⟨n, r⟩ ∈ Le})=⋃T∩NN``{n
∈ nat . ⟨n, r⟩ ∈ Le}
        unfolding RestrictedTo_def by auto moreover
      have ⋃T∩NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}=NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}
        using func1_1_L6(2)[OF R] by blast
      moreover have (T{restricted to}NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}){is
a topology}
        using Top_1_L4 unfolding topology0_def by auto
      ultimately have (NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}){is compact in}(T{restricted
to}NN``{n ∈ nat . ⟨n, r⟩ ∈ Le})
        unfolding Spec_def by force
      then have (NN``{n ∈ nat . ⟨n, r⟩ ∈ Le}){is compact in}(T) us-
ing compact_subspace_imp_compact by auto
      moreover from Acov(1) have (NN``{n ∈ nat . ⟨n, r⟩ ∈ Le})⊆⋃M
by auto
      moreover note Acov(2) ultimately
      obtain 𝔑 where 𝔑:𝔑∈FinPow(M) (NN``{n ∈ nat . ⟨n, r⟩ ∈ Le})⊆⋃𝔑
        unfolding IsCompact_def by blast
      from 𝔑(1) have 𝔑 ∪{V}∈FinPow(M) using V(2) unfolding FinPow_def
by auto moreover
      {
        fix s assume s:s∈A s∉V
```

```
            with V(1) have s∈NNnat by auto
            then have s∈{NNn. n∈nat} using func_imagedef[OF NFun] by
auto
            then obtain n where n:n∈nat s=NNn by auto
            {
              assume ⟨r,n⟩∈Le
              with rr have NNn∈V by auto
              with n(2) s(2) have False by auto
            }
            then have ⟨r,n⟩∉Le by auto
            with rr(1) n(1) have ¬(r≤n) by auto
            then have n≤r using Ord_linear_le[where thesis=⟨n,r⟩∈Le]
nat_into_Ord[OF rr(1)]
              nat_into_Ord[OF n(1)] by auto
            with rr(1) n(1) have ⟨n,r⟩∈Le by auto
            with n(2) have s∈{NNt. t∈{n∈nat. ⟨n,r⟩∈Le}} by auto more-
over
            have {n∈nat. ⟨n,r⟩∈Le}⊆nat by auto
            ultimately have s∈NN{n∈nat. ⟨n,r⟩∈Le} using func_imagedef[OF
NFun]
              by auto
            with 𝔑(2) have s∈⋃𝔑 by auto
          }
          then have A⊆⋃𝔑 ∪ V by auto
          then have A⊆⋃(𝔑 ∪ {V}) by auto ultimately
          have ∃𝔑∈FinPow(M). A⊆⋃𝔑 by auto
        }
        then have ∀M∈Pow(T). A⊆⋃M ⟶ (∃𝔑∈FinPow(M). A⊆⋃𝔑) by auto
moreover
        have ss:A⊆⋃(T) using func1_1_L6(2)[OF R] y by blast ultimately
        have A{is compact in}(T) unfolding IsCompact_def by auto more-
over
        with assms have A{is closed in}(T) unfolding IsKC_def IsCompact_def
by auto ultimately
        have A∈{B∈Pow(⋃T). B{is compact in}(T)∧B{is closed in}(T)} us-
ing ss by auto
        then have {⋃T}∪(⋃T-A)∈({one-point compactification of}T) un-
folding OPCompactification_def
          by auto
        then have {⋃T}∪(⋃T-A)=Interior({⋃T}∪(⋃T-A),{one-point compactification
of}T) using topology0.Top_2_L3 op_comp_is_top
          unfolding topology0_def by auto moreover
        {
          assume x∈A
          with A(4) have x∈NNnat by auto
          then have x∈{NNn. n∈nat} using func_imagedef[OF NFun] by auto
          then obtain n where n∈natNNn=x by auto
          with noy x have False by auto
        }
```

```
        with y have x∈{⋃T}∪(⋃T-A) using x by force ultimately
        have x∈Interior({⋃T}∪(⋃T-A),{one-point compactification of}T)
{⋃T}∪(⋃T-A)∈Pow(⋃({one-point compactification of}T))
          using op_compact_total by auto moreover
        have (∀U∈Pow(⋃({one-point compactification of}T)).  x ∈ Interior(U,{one-point
compactification of}T) ⟶ (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ U))
          using A(2) dom topology0.NetConverges_def[OF _ NET] op_comp_is_top
unfolding topology0_def by auto
        ultimately have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ {⋃T}∪(⋃T-A)
by blast
        then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈{⋃T}∪(⋃T-A)
by auto
        {
          fix s assume AA:⟨r,s⟩∈Le
          with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
          then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm.
Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
            nat_into_Ord by auto
          with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by
auto
          with r_def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈{⋃T}∪(⋃T-A)
by auto
          then have NNs∈{⋃T}∪(⋃T-A) using apply_equality[OF _ NFun]
AA by auto
          with noy have NNs∈(⋃T-A) using AA by auto
          moreover have NNs∈{NNt. t∈nat} using AA by auto
          then have NNs∈NNnat using func_imagedef[OF NFun] by auto
          then have NNs∈A by auto
          ultimately have False by auto
        }
        moreover have r⊆succ(r) using succ_explained by auto
        then have r≤succ(r) using subset_imp_le nat_into_Ord ‹r∈nat›
nat_succI
          by auto
        then have ⟨r,succ(r)⟩∈Le using ‹r∈nat› nat_succI by auto
        ultimately have False by auto
      }
      then have x≠⋃T by auto
      with xy(1) AAA have y∉⋃T x∈⋃T using op_compact_total by auto
      with xy(2) have y:y=⋃T and x:x∈⋃T using op_compact_total by auto
      {
        assume B:∃n∈nat. ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T
        have ⋃T∈({one-point compactification of}T) using open_subspace
by auto
        then have ⋃T=Interior(⋃T,{one-point compactification of}T) us-
ing topology0.Top_2_L3
          unfolding topology0_def using op_comp_is_top by auto
        then have x∈Interior(⋃T,{one-point compactification of}T) us-
ing x by auto moreover
```

```
        have ⋃T∈Pow(⋃({one-point compactification of}T)) using open_subspace(1)
by auto
        ultimately have ∃t∈domain(fst(⟨N, Le⟩)). ∀m∈domain(fst(⟨N, Le⟩)).
⟨t, m⟩ ∈ snd(⟨N, Le⟩) ⟶ fst(⟨N, Le⟩)  m ∈ ⋃T using A(2)
          using topology0.NetConverges_def[OF _ NET] op_comp_is_top un-
folding topology0_def by blast
        then have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ ⋃T using dom
by auto
        then obtain t where t:t∈nat ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ ⋃T
by auto
        from B obtain n where n:n∈nat ∀m∈nat. ⟨n,m⟩∈Le ⟶ Nm=⋃T by
auto
        from t(1) n(1) dir obtain z where z:z∈nat ⟨n,z⟩∈Le ⟨t,z⟩∈Le un-
folding IsDirectedSet_def
          by auto
        from t(2) z(1,3) have Nz∈⋃T by auto moreover
        from n(2) z(1,2) have Nz=⋃T by auto ultimately
        have False using mem_not_refl by auto
      }
      then have reg:∀n∈nat. ∃m∈nat. Nm≠⋃T ∧ ⟨n,m⟩∈Le by auto
      let NN={⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩. n∈nat}
      {
        fix x z assume A1:⟨x, z⟩ ∈ NN
        {
          fix y' assume A2:⟨x,y'⟩∈NN
          with A1 have z=y' by auto
        }
        then have ∀y'. ⟨x,y'⟩∈NN ⟶ z=y' by auto
      }
      then have ∀x z. ⟨x, z⟩ ∈ NN ⟶ (∀y'. ⟨x,y'⟩∈NN ⟶ z=y') by auto
      moreover
      {
        fix n assume as:n∈nat
        with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
        then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧
⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
          nat_into_Ord[of m] by auto
        then have (μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈nat by auto
        then have N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)∈⋃({one-point compactification
of}T) using apply_type[OF A(1)] op_compact_total by auto
        with as have ⟨n,N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)⟩∈nat×⋃({one-point
compactification of}T) by auto
      }
      then have NN∈Pow(nat×⋃({one-point compactification of}T)) by
auto
      ultimately have NFun:NN:nat→⋃({one-point compactification of}T)
unfolding Pi_def function_def domain_def by auto
      {
        fix n assume as:n∈nat
```

```
        with reg obtain m where Nm≠⋃T ∧ ⟨n,m⟩∈Le m∈nat by auto
        then have LI:N(μ i. Ni≠⋃T ∧ ⟨n,i⟩∈Le)≠⋃T ⟨n,μ i. Ni≠⋃T ∧
⟨n,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T ∧ ⟨n,m⟩∈Le m]
          nat_into_Ord[of m] by auto
        then have NNn≠⋃T using apply_equality[OF _ NFun] by auto
      }
      then have noy:∀n∈nat. NNn≠⋃T by auto
      then have ∀n∈nat. NNn∈⋃T using apply_type[OF NFun] op_compact_total
by auto
      then have R:NN:nat→⋃T using func1_1_L1A[OF NFun] by auto
      have dom2:domain(NN)=nat by auto
      then have net2:⟨NN,Le⟩{is a net on}⋃T unfolding IsNet_def using
R dir by auto
      {
        fix U assume U:U⊆⋃T x∈int(U)
        have intT:int(U)∈T using Top_2_L2 by auto
        then have int(U)∈({one-point compactification of}T) unfolding
OPCompactification_def
          by auto
        then have Interior(int(U),{one-point compactification of}T)=int(U)
using topology0.Top_2_L3
          unfolding topology0_def using op_comp_is_top by auto
        with U(2) have x∈Interior(int(U),{one-point compactification
of}T) by auto
        with intT have (∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈int(U)) us-
ing comp op_compact_total by auto
        then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈U us-
ing Top_2_L1 by auto
        {
          fix s assume AA:⟨r,s⟩∈Le
          with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
          then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm.
Nm≠⋃T ∧ ⟨s,m⟩∈Le m]
            nat_into_Ord by auto
          with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by
auto
          with r_def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈U by blast
          then have NNs∈U using apply_equality[OF _ NFun] AA by auto
        }
        then have ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
        with r_def(1) have ∃r∈nat. ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈U by auto
      }
      then have ∀U∈Pow(⋃T). x ∈ int(U)
        ⟶ (∃r∈nat. ∀s∈nat. ⟨r, s⟩ ∈ Le ⟶ NN  s ∈ U) by auto
      then have cony:⟨NN,Le⟩→_N x{in}T using NetConverges_def[OF net2]
x op_comp_is_top
        unfolding topology0_def using xy(2) dom2 by auto
      let A={x}∪NNnat
      {
```

```
        fix M assume Acov:A⊆⋃M M⊆T
        then have x∈⋃M by auto
        then obtain V where V:x∈V V∈M by auto
        with Acov(2) have VT:V∈T by auto
        then have V=int(V) using Top_2_L3 by auto
        with V(1) have x∈int(V) by auto
        with cony VT obtain r where rr:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ NNs∈V
          unfolding NetConverges_def[OF net2, of x] using dom2 y by auto
        have NresFun:restrict(NN,{n∈nat. ⟨n,r⟩∈Le}):{n∈nat. ⟨n,r⟩∈Le}→⋃T
using restrict_fun
          [OF R, of {n∈nat. ⟨n,r⟩∈Le}] by auto
        then have restrict(NN,{n∈nat. ⟨n,r⟩∈Le})∈surj({n∈nat. ⟨n,r⟩∈Le},range(restrict(NN,{
⟨n,r⟩∈Le})))
          using fun_is_surj by auto moreover
        have {n∈nat. ⟨n,r⟩∈Le}⊆nat by auto
        then have {n∈nat. ⟨n,r⟩∈Le}≲nat using subset_imp_lepoll by auto
        ultimately have range(restrict(NN,{n∈nat. ⟨n,r⟩∈Le}))≲{n∈nat.
⟨n,r⟩∈Le} using surj_fun_inv_2 by auto
        moreover
        have {n∈nat. ⟨n,0⟩∈Le}={0} by auto
        then have Finite({n∈nat. ⟨n,0⟩∈Le}) by auto moreover
        {
          fix j assume as:j∈nat Finite({n∈nat. ⟨n,j⟩∈Le})
          {
            fix t assume t∈{n∈nat. ⟨n,succ(j)⟩∈Le}
            then have t∈nat ⟨t,succ(j)⟩∈Le by auto
            then have t≤succ(j) by auto
            then have t⊆succ(j) using le_imp_subset by auto
            then have t⊆j ∪{j} using succ_explained by auto
            then have j∈t∨t⊆j by auto
            then have j∈t∨t≤j using subset_imp_le ‹t∈nat› ‹j∈nat› nat_into_Ord
by auto
            then have j ∪{j}⊆t∨t≤j using ‹t∈nat› ‹j∈nat› nat_into_Ord
unfolding Ord_def
              Transset_def by auto
            then have succ(j)⊆t∨t≤j using succ_explained by auto
            with ‹t⊆succ(j)› have t=succ(j)∨t≤j by auto
            with ‹t∈nat› ‹j∈nat› have t∈{n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}
by auto
          }
          then have {n∈nat. ⟨n,succ(j)⟩∈Le} ⊆{n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}
by auto
          moreover have Finite({n∈nat. ⟨n,j⟩∈Le} ∪ {succ(j)}) using as(2)
Finite_cons
            by auto
          ultimately have Finite({n∈nat. ⟨n,succ(j)⟩∈Le}) using subset_Finite
by auto
        }
        then have ∀j∈nat. Finite({n∈nat. ⟨n,j⟩∈Le}) ⟶ Finite({n∈nat.
```

```
⟨n,succ(j)⟩∈Le})
          by auto
        ultimately have Finite(range(restrict(NN, {n ∈ nat . ⟨n, r⟩ ∈
Le})))
          using lepoll_Finite[of range(restrict(NN, {n ∈ nat . ⟨n, r⟩
∈ Le}))
            {n ∈ nat . ⟨n, r⟩ ∈ Le}] ind_on_nat[OF ‹r∈nat›, where P=λt.
Finite({n∈nat. ⟨n,t⟩∈Le})] by auto
        then have Finite((restrict(NN, {n ∈ nat . ⟨n, r⟩ ∈ Le})){n ∈
nat . ⟨n, r⟩ ∈ Le}) using range_image_domain[OF NresFun]
          by auto
        then have Finite(NN{n ∈ nat . ⟨n, r⟩ ∈ Le}) using restrict_image
by auto
        then have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is in the spectrum of}(λT.
(⋃T){is compact in}T) using compact_spectrum by auto
        moreover have ⋃(T{restricted to}NN{n ∈ nat . ⟨n, r⟩ ∈ Le})=⋃T∩NN{n
∈ nat . ⟨n, r⟩ ∈ Le}
          unfolding RestrictedTo_def by auto moreover
        have ⋃T∩NN{n ∈ nat . ⟨n, r⟩ ∈ Le}=NN{n ∈ nat . ⟨n, r⟩ ∈ Le}
          using func1_1_L6(2)[OF R] by blast
        moreover have (T{restricted to}NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is
a topology}
          using Top_1_L4 unfolding topology0_def by auto
        ultimately have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is compact in}(T{restricted
to}NN{n ∈ nat . ⟨n, r⟩ ∈ Le})
          unfolding Spec_def by force
        then have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le}){is compact in}(T) using
compact_subspace_imp_compact by auto
        moreover from Acov(1) have (NN{n ∈ nat . ⟨n, r⟩ ∈ Le})⊆⋃M by
auto
        moreover note Acov(2) ultimately
        obtain 𝔑 where 𝔑:𝔑∈FinPow(M) (NN{n ∈ nat . ⟨n, r⟩ ∈ Le})⊆⋃𝔑
          unfolding IsCompact_def by blast
        from 𝔑(1) have 𝔑 ∪{V}∈FinPow(M) using V(2) unfolding FinPow_def
by auto moreover
        {
          fix s assume s:s∈A s∉V
          with V(1) have s∈NNnat by auto
          then have s∈{NNn. n∈nat} using func_imagedef[OF NFun] by auto
          then obtain n where n:n∈nat s=NNn by auto
          {
            assume ⟨r,n⟩∈Le
            with rr have NNn∈V by auto
            with n(2) s(2) have False by auto
          }
          then have ⟨r,n⟩∉Le by auto
          with rr(1) n(1) have ¬(r≤n) by auto
          then have n≤r using Ord_linear_le[where thesis=⟨n,r⟩∈Le] nat_into_Ord[OF
rr(1)]
```

```
          nat_into_Ord[OF n(1)] by auto
        with rr(1) n(1) have ⟨n,r⟩∈Le by auto
        with n(2) have s∈{NNt. t∈{n∈nat. ⟨n,r⟩∈Le}} by auto more-
over
        have {n∈nat. ⟨n,r⟩∈Le}⊆nat by auto
        ultimately have s∈NN{n∈nat. ⟨n,r⟩∈Le} using func_imagedef[OF
NFun]
          by auto
        with 𝔑(2) have s∈⋃𝔑 by auto
      }
      then have A⊆⋃𝔑 ∪ V by auto
      then have A⊆⋃(𝔑 ∪ {V}) by auto ultimately
      have ∃𝔑∈FinPow(M). A⊆⋃𝔑 by auto
    }
    then have ∀M∈Pow(T). A⊆⋃M ⟶ (∃𝔑∈FinPow(M). A⊆⋃𝔑) by auto
moreover
    have ss:A⊆⋃(T) using func1_1_L6(2)[OF R] x by blast ultimately
    have A{is compact in}(T) unfolding IsCompact_def by auto more-
over
    with assms have A{is closed in}(T) unfolding IsKC_def IsCompact_def
by auto ultimately
    have A∈{B∈Pow(⋃T). B{is compact in}(T)∧B{is closed in}(T)} us-
ing ss by auto
    then have {⋃T}∪(⋃T-A)∈({one-point compactification of}T) un-
folding OPCompactification_def
      by auto
    then have {⋃T}∪(⋃T-A)=Interior({⋃T}∪(⋃T-A),{one-point compactification
of}T) using topology0.Top_2_L3 op_comp_is_top
      unfolding topology0_def by auto moreover
    {
      assume y∈A
      with A(4) have y∈NNnat by auto
      then have y∈{NNn. n∈nat} using func_imagedef[OF NFun] by auto
      then obtain n where n∈natNNn=y by auto
      with noy y have False by auto
    }
    with y have y∈{⋃T}∪(⋃T-A) by force ultimately
    have y∈Interior({⋃T}∪(⋃T-A),{one-point compactification of}T)
{⋃T}∪(⋃T-A)∈Pow(⋃({one-point compactification of}T))
      using op_compact_total by auto moreover
    have (∀U∈Pow(⋃({one-point compactification of}T)). y ∈ Interior(U,{one-point
compactification of}T) ⟶ (∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ U))
      using A(3) dom topology0.NetConverges_def[OF _ NET] op_comp_is_top
unfolding topology0_def by auto
    ultimately have ∃t∈nat. ∀m∈nat. ⟨t, m⟩ ∈ Le ⟶ N  m ∈ {⋃T}∪(⋃T-A)
by blast
    then obtain r where r_def:r∈nat ∀s∈nat. ⟨r,s⟩∈Le ⟶ Ns∈{⋃T}∪(⋃T-A)
by auto
    {
```

```
        fix s assume AA:⟨r,s⟩∈Le
        with reg obtain m where Nm≠⋃T ⟨s,m⟩∈Le by auto
        then have ⟨s,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using LeastI[of λm. Nm≠⋃T
∧ ⟨s,m⟩∈Le m]
          nat_into_Ord by auto
        with AA have ⟨r,μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le⟩∈Le using le_trans by
auto
        with r_def(2) have N(μ i. Ni≠⋃T ∧ ⟨s,i⟩∈Le)∈{⋃T}∪(⋃T-A) by
auto
        then have NNs∈{⋃T}∪(⋃T-A) using apply_equality[OF _ NFun] AA
by auto
        with noy have NNs∈(⋃T-A) using AA by auto
        moreover have NNs∈{NNt. t∈nat} using AA by auto
        then have NNs∈NNnat using func_imagedef[OF NFun] by auto
        then have NNs∈A by auto
        ultimately have False by auto
      }
      moreover have r⊆succ(r) using succ_explained by auto
      then have r≤succ(r) using subset_imp_le nat_into_Ord ‹r∈nat› nat_succI
        by auto
      then have ⟨r,succ(r)⟩∈Le using ‹r∈nat› nat_succI by auto
      ultimately have False by auto
    }
    ultimately have False by auto
  }
  then have ∀N x y. N:nat→(⋃{one-point compactification of}T) ∧ (⟨N,Le⟩→N
x{in}({one-point compactification of}T))
    ∧ (⟨N,Le⟩→N y{in}({one-point compactification of}T)) ⟶ x=y by auto
  then show thesis unfolding IsUS_def by auto
qed
```

In the one-point compactification of an anti-compact space, ever subspace that contains the infinite point is compact.

```
theorem (in topology0) anti_comp_imp_OP_inf_comp:
  assumes T{is anti-compact} A⊆⋃({one-point compactification of}T) ⋃T∈A
  shows A{is compact in}({one-point compactification of}T)
proof-
  {
    fix M assume M:M⊆({one-point compactification of}T) A⊆⋃M
    with assms(3) obtain U where U:⋃T∈U U∈M by auto
    with M(1) obtain K where K:K{is compact in}T K{is closed in}T U={⋃T}∪(⋃T-K)
      unfolding OPCompactification_def using mem_not_refl[of ⋃T] by auto
    from K(1) have K{is compact in}(T{restricted to}K) using compact_imp_compact_subspace
Compact_is_card_nat
      by auto
    moreover have ⋃(T{restricted to}K)=⋃T∩K unfolding RestrictedTo_def
by auto
    with K(1) have ⋃(T{restricted to}K)=K unfolding IsCompact_def by
auto ultimately
```

```
    have (⋃(T{restricted to}K)){is compact in}(T{restricted to}K) by
auto
    with assms(1) have K{is in the spectrum of}(λT. (⋃T){is compact
in}T) unfolding IsAntiComp_def
      antiProperty_def using K(1) unfolding IsCompact_def by auto
    then have finK:Finite(K) using compact_spectrum by auto
    from assms(2) have A-U⊆(⋃T ∪{⋃T}) -U using op_compact_total by
auto
    with K(3) have A-U⊆K by auto
    with finK have Finite(A-U) using subset_Finite by auto
    then have (A-U){is in the spectrum of}(λT. (⋃T){is compact in}T)
using compact_spectrum by auto moreover
    have ⋃(({one-point compactification of}T){restricted to}(A-U))=A-U
unfolding RestrictedTo_def using assms(2) K(3)
      op_compact_total by auto moreover
    have (({one-point compactification of}T){restricted to}(A-U)){is a
topology} using topology0.Top_1_L4
      op_comp_is_top unfolding topology0_def by auto
    ultimately have (A-U){is compact in}(({one-point compactification
of}T){restricted to}(A-U))
      unfolding Spec_def by auto
    then have (A-U){is compact in}({one-point compactification of}T)
using compact_subspace_imp_compact by auto
    moreover have A-U⊆⋃M using M(2) by auto moreover
    note M(1) ultimately obtain N where N:N∈FinPow(M) A-U⊆⋃N unfold-
ing IsCompact_def by blast
    from N(1) U(2) have N ∪{U}∈FinPow(M) unfolding FinPow_def by auto
moreover
    from N(2) have A⊆⋃(N ∪{U}) by auto
    ultimately have ∃R∈FinPow(M). A⊆⋃R by auto
  }
  then show thesis using op_compact_total assms(2) unfolding IsCompact_def
by auto
qed
```

As a last result in this section, the one-point compactification of our topology is not a KC space.

```
theorem extension_pow_OP_not_KC:
  shows ¬(({one-point compactification of}(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S.
S∈(CoCountable csucc(nat))-{0}})){is KC})
proof
  have noE:csucc(nat)≠0 using Ord_0_lt_csucc[OF Ord_nat] by auto
  let T=(Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}})
  assume ass:({one-point compactification of}T){is KC}
  from extension_pow_notDiscrete have {csucc(nat)} ∉ (Pow(csucc(nat))
∪ {{csucc(nat)} ∪ S . S ∈ (CoCountable csucc(nat)) - {0}})
    by auto
  {
    assume csucc(nat)=csucc(nat)∪{csucc(nat)} moreover
```

```
    have csucc(nat)∈csucc(nat)∪{csucc(nat)} by auto
    ultimately have csucc(nat)∈csucc(nat) by auto
    then have False using mem_not_refl by auto
  }
  then have dist:csucc(nat)≠csucc(nat)∪{csucc(nat)} by blast
  {
    assume {csucc(nat)}∈({one-point compactification of}(Pow(csucc(nat))
∪ {{csucc(nat)} ∪ S . S ∈ (CoCountable csucc(nat)) - {0}}))
    then have {csucc(nat)}∈{{⋃T}∪((⋃T)-K). K∈{B∈Pow(⋃T). B{is compact
in}T ∧ B{is closed in}T}}
      unfolding OPCompactification_def using extension_pow_notDiscrete
by auto
    then obtain K where {csucc(nat)}={⋃T}∪((⋃T)-K) by auto moreover
    have ⋃T∈{⋃T}∪((⋃T)-K) by auto
    ultimately have ⋃T∈{csucc(nat)} by auto
    with dist have False using extension_pow_union by auto
  }
  then have {csucc(nat)}∉({one-point compactification of}T) by auto more-
over
  have ⋃({one-point compactification of}T)-(⋃({one-point compactification
of}T)-{csucc(nat)})={csucc(nat)} using extension_pow_union
    topology0.op_compact_total unfolding topology0_def using extension_pow_top
by auto ultimately
  have di:⋃({one-point compactification of}T)-(⋃({one-point compactification
of}T)-{csucc(nat)})∉({one-point compactification of}T) by auto
  {
    assume (⋃({one-point compactification of}T)-{csucc(nat)}){is closed
in}({one-point compactification of}T)
    then have ⋃({one-point compactification of}T)-(⋃({one-point compactification
of}T)-{csucc(nat)})∈({one-point compactification of}T) unfolding IsClosed_def
by auto
    with di have False by auto
  }
  then have n:¬((⋃({one-point compactification of}T)-{csucc(nat)}){is
closed in}({one-point compactification of}T)) by auto moreover
  from dist have ⋃T∈(⋃({one-point compactification of}T)-{csucc(nat)})
using topology0.op_compact_total unfolding topology0_def using extension_pow_top
    extension_pow_union by auto
  then have (⋃({one-point compactification of}T)-{csucc(nat)}){is compact
in}({one-point compactification of}T) using topology0.anti_comp_imp_OP_inf_comp[of
T
    (⋃({one-point compactification of}T)-{csucc(nat)})] unfolding topology0_def
using extension_pow_antiCompact extension_pow_top by auto
  with ass have (⋃({one-point compactification of}T)-{csucc(nat)}){is
closed in}({one-point compactification of}T) unfolding IsKC_def by auto
  with n show False by auto
qed
```

In conclusion, $US \nRightarrow KC$.

89.8 Other types of properties

In this section we will define new properties that aren't defined as anti-properties and that are not separation axioms. In some cases we will consider their anti-properties.

89.9 Definitions

A space is called perfect if it has no isolated points. This definition may vary in the literature to similar, but not equivalent definitions.

definition
  IsPerf (_ {is perfect}) **where**
  T{is perfect} ≡ ∀x∈⋃T. {x}∉T

An anti-perfect space is called scattered.

definition
  IsScatt (_ {is scattered}) **where**
  T{is scattered} ≡ T{is anti-}IsPerf

A topological space with two disjoint dense subspaces is called resolvable.

definition
  IsRes (_ {is resolvable}) **where**
  T{is resolvable} ≡ ∃U∈Pow(⋃T). ∃V∈Pow(⋃T). Closure(U,T)=⋃T ∧ Closure(V,T)=⋃T ∧ U∩V=0

A topological space where every dense subset is open is called submaximal.

definition
  IsSubMax (_ {is submaximal}) **where**
  T{is submaximal} ≡ ∀U∈Pow(⋃T). Closure(U,T)=⋃T ⟶ U∈T

A subset of a topological space is nowhere-dense if the interior of its closure is empty.

definition
  IsNowhereDense (_ {is nowhere dense in} _) **where**
  A{is nowhere dense in}T ≡ A⊆⋃T ∧ Interior(Closure(A,T),T)=0

A topological space is then a Luzin space if every nowhere-dense subset is countable.

definition
  IsLuzin (_ {is luzin}) **where**
  T{is luzin} ≡ ∀A∈Pow(⋃T). (A{is nowhere dense in}T) ⟶ A≲nat

An also useful property is local-connexion.

definition
  IsLocConn (_{is locally-connected}) **where**

T{is locally-connected} ≡ T{is locally}(λT. λB. ((T{restricted to}B){is connected}))

An SI-space is an anti-resolvable perfect space.

definition
IsAntiRes (_{is anti-resolvable}) **where**
T{is anti-resolvable} ≡ T{is anti-}IsRes

definition
IsSI (_{is Strongly Irresolvable}) **where**
T{is Strongly Irresolvable} ≡ (T{is anti-resolvable}) ∧ (T{is perfect})

89.10 First examples

Firstly, we need to compute the spectrum of the being perfect.

lemma spectrum_perfect:
shows (A{is in the spectrum of}IsPerf) ⟷ A=0
proof
assume A{is in the spectrum of}IsPerf
then have Pow(A){is perfect} **unfolding** Spec_def **using** Pow_is_top **by** auto
then have ∀b∈A. {b}∉Pow(A) **unfolding** IsPerf_def **by** auto
then show A=0 **by** auto
next
assume A:A=0
{
fix T **assume** T:T{is a topology} ⋃T≈A
with T(2) A **have** ⋃T≈0 **by** auto
then have ⋃T=0 **using** eqpoll_0_is_0 **by** auto
then have T{is perfect} **unfolding** IsPerf_def **by** auto
}
then show A{is in the spectrum of}IsPerf **unfolding** Spec_def **by** auto
qed

The discrete space is clearly scattered:

lemma pow_is_scattered:
shows Pow(A){is scattered}
proof-
{
fix B **assume** B:B⊆⋃Pow(A) (Pow(A){restricted to}B){is perfect}
from B(1) **have** Pow(A){restricted to}B=Pow(B) **unfolding** RestrictedTo_def **by** blast
with B(2) **have** Pow(B){is perfect} **by** auto
then have ∀b∈B. {b}∉Pow(B) **unfolding** IsPerf_def **by** auto
then have B=0 **by** auto
}
then show thesis **using** spectrum_perfect **unfolding** IsScatt_def antiProperty_def **by** auto

qed

The trivial topology is perfect, if it is defined over a set with more than one point.

```
lemma trivial_is_perfect:
  assumes ∃x y. x∈X ∧ y∈X ∧ x≠y
  shows {0,X}{is perfect}
proof-
  {
    fix r assume {r}∈{0,X}
    then have X={r} by auto
    with assms have False by auto
  }
  then show thesis unfolding IsPerf_def by auto
qed
```

The trivial topology is resolvable, if it is defined over a set with more than one point.

```
lemma trivial_is_resolvable:
  assumes ∃x y. x∈X ∧ y∈X ∧ x≠y
  shows {0,X}{is resolvable}
proof-
  from assms obtain x y where xy:x∈X y∈X x≠y by auto
  {
    fix A assume A:A{is closed in}{0,X} A⊆X
    then have X-A∈{0,X} unfolding IsClosed_def by auto
    then have X-A=0∨X-A=X by auto
    with A(2) have A=X∨X-A=X by auto moreover
    {
      assume X-A=X
      then have X-(X-A)=0 by auto
      with A(2) have A=0 by auto
    }
    ultimately have A=X∨A=0 by auto
    then have A=0∨A=X by auto
  }
  then have cl:∀A∈Pow(X). A{is closed in}{0,X} ⟶ A=0∨A=X by auto
  from xy(3) have {x}∩{y}=0 by auto moreover
  {
    have {X}{is a partition of}X using indiscrete_partition xy(1) by
auto
    then have top:topology0(PTopology X {X}) using topology0_ptopology
by auto
    have X≠0 using xy(1) by auto
    then have (PTopology X {X})={0,X} using indiscrete_ptopology[of X]
by auto
    with top have top0:topology0({0,X}) by auto
    then have x∈Closure({x},{0,X}) using topology0.cl_contains_set xy(1)
by auto moreover
```

```
    have Closure({x},{0,X}) {is closed in}{0,X} using topology0.cl_is_closed
top0 xy(1) by auto
    moreover note cl
    moreover have Closure({x},{0,X})⊆X using topology0.Top_3_L11(1)
top0 xy(1) by auto
    ultimately have Closure({x},{0,X})=X by auto
  }
  moreover
  {
    have {X}{is a partition of}X using indiscrete_partition xy(1) by
auto
    then have top:topology0(PTopology X {X}) using topology0_ptopology
by auto
    have X≠0 using xy(1) by auto
    then have (PTopology X {X})={0,X} using indiscrete_ptopology[of X]
by auto
    with top have top0:topology0({0,X}) by auto
    then have y∈Closure({y},{0,X}) using topology0.cl_contains_set xy(2)
by auto moreover
    have Closure({y},{0,X}) {is closed in}{0,X} using topology0.cl_is_closed
top0 xy(2) by auto
    moreover note cl
    moreover have Closure({y},{0,X})⊆X using topology0.Top_3_L11(1)
top0 xy(2) by auto
    ultimately have Closure({y},{0,X})=X by auto
  }
  ultimately show thesis using xy(1,2) unfolding IsRes_def by auto
qed
```

The spectrum of Luzin spaces is the class of countable sets, so there are lots of examples of Luzin spaces.

```
lemma spectrum_Luzin:
  shows (A{is in the spectrum of}IsLuzin) ⟷ A≲nat
proof
  assume A:A{is in the spectrum of}IsLuzin
  {
    assume A=0
    then have A≲nat using empty_lepollI by auto
  }
  moreover
  {
    assume A≠0
    then obtain x where x:x∈A by auto
    {
      fix M assume M⊆{0,{x},A}
      then have ⋃M∈{0,{x},A} using x by blast
    }
    moreover
    {
```

```
      fix U V assume U∈{0,{x},A} V∈{0,{x},A}
      then have U∩V∈{0,{x},A} by auto
    }
    ultimately have top:{0,{x},A}{is a topology} unfolding IsATopology_def
by auto
    moreover have tot:⋃{0,{x},A}=A using x by auto
    moreover note A ultimately have luz:{0,{x},A}{is luzin} unfolding
Spec_def by auto
    moreover have {x}∈{0,{x},A} by auto
    then have ((⋃{0,{x},A})-{x}){is closed in}{0,{x},A} using topology0.Top_3_L9
      unfolding topology0_def using top by blast
    then have (A-{x}){is closed in}{0,{x},A} using tot by auto
    then have Closure(A-{x},{0,{x},A})=A-{x} using tot top topology0.Top_3_L8[of
{0,{x},A}]
      unfolding topology0_def by auto
    then have B:Interior(Closure(A-{x},{0,{x},A}),{0,{x},A})=Interior(A-{x},{0,{x},A})
by auto
    then have C:Interior(Closure(A-{x},{0,{x},A}),{0,{x},A})⊆A-{x} us-
ing top topology0.Top_2_L1
      unfolding topology0_def by auto
    then have D:Interior(Closure(A-{x},{0,{x},A}),{0,{x},A})∈{0,{x},A}
using topology0.Top_2_L2
      unfolding topology0_def using top by auto
    from x have ¬(A⊆A-{x}) by auto
    with C D have Interior(Closure(A-{x},{0,{x},A}),{0,{x},A})=0 by auto
    then have (A-{x}){is nowhere dense in}{0,{x},A} unfolding IsNowhereDense_def
using tot
      by auto
    with luz have A-{x}≲nat unfolding IsLuzin_def using tot by auto
    then have U1:A-{x}≺csucc(nat) using Card_less_csucc_eq_le[OF Card_nat]
by auto
    have {x}≈1 using singleton_eqpoll_1 by auto
    then have {x}≺nat using n_lesspoll_nat eq_lesspoll_trans by auto
    then have {x}≲nat using lesspoll_imp_lepoll by auto
    then have U2:{x}≺csucc(nat) using Card_less_csucc_eq_le[OF Card_nat]
by auto
    with U1 have U:(A-{x})∪{x}≺csucc(nat) using less_less_imp_un_less[OF
_ _ InfCard_csucc[OF InfCard_nat]]
      by auto
    have (A-{x})∪{x}=A using x by auto
    with U have A≺csucc(nat) by auto
    then have A≲nat using Card_less_csucc_eq_le[OF Card_nat] by auto
  }
  ultimately
  show A≲nat by auto
next
  assume A:A≲nat
  {
    fix T assume T:T{is a topology} ⋃T≈A
```

```
    {
      fix B assume B⊆⋃T B{is nowhere dense in}T
      then have B≲⋃T using subset_imp_lepoll by auto
      with T(2) have B≲A using lepoll_eq_trans by auto
      with A have B≲nat using lepoll_trans by blast
    }
    then have ∀B∈Pow(⋃T). (B{is nowhere dense in}T) ⟶ B≲nat by auto
    then have T{is luzin} unfolding IsLuzin_def by auto
  }
  then show A{is in the spectrum of}IsLuzin unfolding Spec_def by auto
qed
```

89.11 Structural results

Every resolvable space is also perfect.

```
theorem (in topology0) resolvable_imp_perfect:
  assumes T{is resolvable}
  shows T{is perfect}
proof-
  {
    assume ¬(T{is perfect})
    then obtain x where x:x∈⋃T {x}∈T unfolding IsPerf_def by auto
    then have cl:(⋃T-{x}){is closed in}T using Top_3_L9 by auto
    from assms obtain U V where UV:U⊆⋃T V⊆⋃T cl(U)=⋃T cl(V)=⋃T U∩V=0
unfolding IsRes_def by auto
    {
      fix W assume x∉W W⊆⋃T
      then have W⊆⋃T-{x} by auto
      then have cl(W)⊆⋃T-{x} using cl Top_3_L13 by auto
      with x(1) have ¬(⋃T⊆cl(W)) by auto
      then have ¬(cl(W)=⋃T) by auto
    }
    with UV have False by auto
  }
  then show thesis by auto
qed
```

The spectrum of being resolvable follows:

```
corollary spectrum_resolvable:
  shows (A{is in the spectrum of}IsRes) ⟷ A=0
proof
  assume A:A{is in the spectrum of}IsRes
  have ∀T. T{is a topology} ⟶ IsRes(T) ⟶ IsPerf(T) using topology0.resolvable_imp_perfe

    unfolding topology0_def by auto
  with A have A{is in the spectrum of}IsPerf using P_imp_Q_spec_inv[of
IsRes IsPerf] by auto
  then show A=0 using spectrum_perfect by auto
next
```

```
  assume A:A=0
  {
    fix T assume T:T{is a topology} ⋃T≈A
    with T(2) A have ⋃T≈0 by auto
    then have ⋃T=0 using eqpoll_0_is_0 by auto
    then have Closure(0,T)=⋃T using topology0.Top_3_L2 T(1)
      topology0.Top_3_L8 unfolding topology0_def by auto
    then have T{is resolvable} unfolding IsRes_def by auto
  }
  then show A{is in the spectrum of}IsRes unfolding Spec_def by auto
qed
```

The cofinite space over ℕ is a T_1, perfect and luzin space.

```
theorem cofinite_nat_perfect:
  shows (CoFinite nat){is perfect}
proof-
  {
    fix x assume x:x∈⋃(CoFinite nat) {x}∈(CoFinite nat)
    then have xn:x∈nat using union_cocardinal unfolding Cofinite_def
by auto
    with x(2) have nat-{x}≺nat unfolding Cofinite_def CoCardinal_def
by auto
    moreover have Finite({x}) by auto
    then have {x}≺nat unfolding Finite_def using n_lesspoll_nat eq_lesspoll_trans
by auto
    ultimately have (nat-{x})∪{x}≺nat using less_less_imp_un_less[OF
_ _ InfCard_nat] by auto
    moreover have (nat-{x})∪{x}=nat using xn by auto
    ultimately have False by auto
  }
  then show thesis unfolding IsPerf_def by auto
qed

theorem cofinite_nat_luzin:
  shows (CoFinite nat){is luzin}
proof-
  have nat{is in the spectrum of}IsLuzin using spectrum_Luzin by auto
moreover
  have ⋃(CoFinite nat)=nat using union_cocardinal unfolding Cofinite_def
by auto
  moreover have (CoFinite nat){is a topology} unfolding Cofinite_def
using CoCar_is_topology[OF InfCard_nat]
    by auto
  ultimately show thesis unfolding Spec_def by auto
qed
```

The cocountable topology on $\mathbb{N}^+$ or `csucc(nat)` is also T_1, perfect and luzin; but defined on a set not in the spectrum.

```
theorem cocountable_csucc_nat_perfect:
```

```
  shows (CoCountable csucc(nat)){is perfect}
proof-
  have noE:csucc(nat)≠0 using lt_csucc[OF Ord_nat] by auto
  {
    fix x assume x:x∈⋃(CoCountable csucc(nat)) {x}∈(CoCountable csucc(nat))
    then have xn:x∈csucc(nat) using union_cocardinal noE unfolding Cocountable_def
by auto
    with x(2) have csucc(nat)-{x}≺csucc(nat) unfolding Cocountable_def
CoCardinal_def by auto
    moreover have Finite({x}) by auto
    then have {x}≺nat unfolding Finite_def using n_lesspoll_nat eq_lesspoll_trans
by auto
    then have {x}≲nat using lesspoll_imp_lepoll by auto
    then have {x}≺csucc(nat) using Card_less_csucc_eq_le[OF Card_nat]
by auto
    ultimately have (csucc(nat)-{x})∪{x}≺csucc(nat) using less_less_imp_un_less[OF
_ _ InfCard_csucc[OF InfCard_nat]] by auto
    moreover have (csucc(nat)-{x})∪{x}=csucc(nat) using xn by auto
    ultimately have False by auto
  }
  then show thesis unfolding IsPerf_def by auto
qed

theorem cocountable_csucc_nat_luzin:
  shows (CoCountable csucc(nat)){is luzin}
proof-
  have noE:csucc(nat)≠0 using lt_csucc[OF Ord_nat] by auto
  {
    fix B assume B:B∈Pow(⋃(CoCountable csucc(nat))) B{is nowhere dense
in}(CoCountable csucc(nat)) ¬(B≲nat)
    from B(1) have B⊆csucc(nat) using union_cocardinal noE unfolding
Cocountable_def by auto moreover
    from B(3) have ¬(B≺csucc(nat)) using Card_less_csucc_eq_le[OF Card_nat]
by auto ultimately
    have Closure(B,CoCountable csucc(nat))=csucc(nat) using closure_set_cocardinal
noE unfolding Cocountable_def by auto
    then have Interior(Closure(B,CoCountable csucc(nat)),CoCountable
csucc(nat))=Interior(csucc(nat),CoCountable csucc(nat)) by auto
    with B(2) have 0=Interior(csucc(nat),CoCountable csucc(nat)) un-
folding IsNowhereDense_def by auto moreover
    have csucc(nat)-csucc(nat)=0 by auto
    then have csucc(nat)-csucc(nat)≺csucc(nat) using empty_lepollI Card_less_csucc_eq_le[OF
Card_nat] by auto
    then have Interior(csucc(nat),CoCountable csucc(nat))=csucc(nat)
using interior_set_cocardinal noE unfolding Cocountable_def by auto
    ultimately have False using noE by auto
  }
  then have ∀B∈Pow(⋃(CoCountable csucc(nat))). (B{is nowhere dense in}(CoCountable
csucc(nat))) ⟶ B≲nat by auto
```

```
  then show thesis unfolding IsLuzin_def by auto
qed
```

The existence of T_2, uncountable, perfect and luzin spaces is unprovable in *ZFC*. It is related to the *CH* and Martin's axiom.

```
end
```

90 Ultrafilters

```
theory Ultrafilter_ZF imports Topology_ZF_4 Topology_ZF_examples
begin
```

This theory deals with ultrafilters; which is a maximal filter.

```
definition IsUltrafilter (_{is an ultrafilter on}_ 80)
  where 𝔉{is an ultrafilter on}X ≡ (𝔉{is a filter on}X) ∧ (∀𝔊. (𝔊{is
a filter on}X) ⟶ (𝔉 ⊆ 𝔊 ⟶ 𝔊 = 𝔉))
```

Every set or its opposite is in an ultrafilter

```
lemma set_ultrafilter:
  assumes A∈Pow(X) 𝔉{is an ultrafilter on}X
  shows A∈𝔉 ∨ (X-A)∈𝔉
proof-
  from assms(2) have filter:𝔉{is a filter on}X unfolding IsUltrafilter_def
by auto
  {
    assume a0:A≠0
    {
      assume a:A∉𝔉 X-A∉𝔉
      then obtain 𝔊 where g:𝔊{is a filter on}X A ∈ 𝔊 𝔉 ⊆ 𝔊
        using extend_filter[OF assms(1) _ a0, of 𝔉] filter by auto
      with assms(2) have A∈𝔉 unfolding IsUltrafilter_def by auto
      with a have False by auto
    }
    then have A∈𝔉 ∨ (X-A)∈𝔉 by auto
  } moreover
  {
    assume A=0
    then have X-A=X by auto
    then have X-A∈𝔉 using filter unfolding IsFilter_def by auto
    then have A∈𝔉 ∨ (X-A)∈𝔉 by auto
  } ultimately
  show A∈𝔉 ∨ (X-A)∈𝔉 by blast
qed
```

It contains only one of them

```
lemma only_set_or_opposite:
  assumes A∈Pow(X) 𝔉{is a filter on}X A∈𝔉
  shows (X-A)∉𝔉
```

```
proof
  assume X-A∈𝔉
  with assms have A∩(X-A)∈𝔉 using is_filter_def_split(4) by auto
  moreover have A∩(X-A) = 0 by auto
  ultimately have 0∈𝔉 by auto
  then show False using assms(2) is_filter_def_split(1) by auto
qed
```

If a filter contains a set or its opposite, it is in an ultrafilter

```
lemma set_ultrafilter_equiv:
  assumes 𝔉{is a filter on}X ∀A∈Pow(X). A∈𝔉 ∨ (X-A)∈𝔉
  shows 𝔉{is an ultrafilter on}X unfolding IsUltrafilter_def
proof(safe)
  from assms(1) show 𝔉{is a filter on}X.
  {
    fix 𝔊 assume g:𝔊{is a filter on}X 𝔉 ⊆ 𝔊
    {
      assume 𝔊≠𝔉
      with g(2) obtain h where h:h:𝔊 h∉𝔉 by auto
      from h(1) g(1) have h∈Pow(X) h≠0 unfolding IsFilter_def by auto
      with assms(2) h(2) have X-h∈𝔉 by auto
      with g(2) have X-h:𝔊 by auto
      with h(1) g(1) have (X-h)∩h∈𝔊 unfolding IsFilter_def by auto more-
over
      have (X-h)∩h=0 by auto ultimately
      have 0∈𝔊 by auto
      with g(1) have False unfolding IsFilter_def by auto
    }
    then show 𝔊=𝔉 by auto
  }
qed
```

Filters of containing a point, are ultrafilters

```
lemma include_point_ultrafilter:
  assumes x∈X
  shows {A∈Pow(X). x∈A}{is an ultrafilter on}X
proof(rule set_ultrafilter_equiv)
  show {A ∈ Pow(X) . x ∈ A} {is a filter on} X unfolding IsFilter_def
    apply safe using assms apply simp by auto
  {
    fix A assume A:A∈Pow(X)
    have x:A ∨ x∉A by auto
    with assms have x∈A ∨ x∈X-A by auto
    with A(1) have A∈{A ∈ Pow(X) . x ∈ A} ∨ X-A: {A ∈ Pow(X) . x ∈ A}
by auto
  }
  then show ∀A∈Pow(X). A ∈ {A ∈ Pow(X) . x ∈ A} ∨ X - A ∈ {A ∈ Pow(X)
. x ∈ A} by auto
qed
```

An ultrafilter that has no singleton sets, does not have finite sets

```
lemma ultrafilter_finite_set:
  assumes 𝔉{is an ultrafilter on}X ∀x∈X. {x}∉𝔉 A∈FinPow(X)
  shows A∉𝔉
proof(rule FinPow_induct[of λq. q∉𝔉])
  from assms(1) show 0∉𝔉 unfolding IsUltrafilter_def IsFilter_def by
auto
  show A∈FinPow(X) using assms(3).
  {
    fix B assume b:B∈FinPow(X) B∉𝔉
    {
      fix s assume s:s∈X
      {
        assume cont:B∪{s}∈𝔉
        {
          assume B=0
          then have B∪{s} = {s} by auto
          with s assms(2) cont have False by auto
        } moreover
        {
          assume B≠0
          with b have X-B ∈𝔉 using assms(1) set_ultrafilter[of B X 𝔉]
            unfolding FinPow_def by auto moreover
          from assms(1,2) s have X-{s}∈𝔉 using set_ultrafilter[of {s}
X 𝔉]
            by auto
          ultimately have (X-B)∩(X-{s}) ∈𝔉 using assms(1)
            unfolding IsUltrafilter_def IsFilter_def by auto moreover
          {
            fix q assume q∈(X-B)∩(X-{s})
            then have q:X q∉B q∉{s} by auto
            then have q∈X-(B∪{s}) by auto
          }
          then have (X-B)∩(X-{s}) ⊆ X-(B∪{s}) by auto moreover
          have X-(B∪{s})∈Pow(X) by auto
          ultimately have X-(B∪{s}) ∈𝔉 using assms(1) unfolding IsFilter_def
IsUltrafilter_def
            by blast
          with cont have (X-(B∪{s}))∩(B∪{s}): 𝔉 using assms(1)
            unfolding IsFilter_def IsUltrafilter_def by auto moreover
          {
            fix q assume q∈(X-(B∪{s}))∩(B∪{s})
            then have False by auto
          }
          then have (X-(B∪{s}))∩(B∪{s}) = 0 by auto
          ultimately have 0: 𝔉 by auto
          with assms(1) have False unfolding IsUltrafilter_def IsFilter_def
by auto
        }
```

```
      ultimately have False by auto
    }
    then have B∪{s}∉𝔉 by auto
  }
  then have ∀q∈X. B ∪ {q} ∉ 𝔉 by auto
}
then show ∀A∈FinPow(X). A ∉ 𝔉 ⟶ (∀q∈X. A ∪ {q} ∉ 𝔉) by auto
qed
```

The cofinite filter plays an important role in ultrafilters.

```
corollary ultrafilter_contains_cofinite:
  assumes 𝔉{is an ultrafilter on}X ∀x∈X. {x}∉𝔉
  shows (CoFinite(X))-{0} ⊆ 𝔉
proof
  fix S assume S∈(CoFinite(X))-{0}
  then have s:S ∈ Pow(X) X - S ≺ nat S≠0 unfolding CoCardinal_def Cofinite_def
by auto
  then have Finite(X-S) using lesspoll_nat_is_Finite by auto
  then have X-S∈FinPow(X) unfolding FinPow_def by auto
  with ultrafilter_finite_set assms have X-S∉𝔉 by auto
  with assms(1) show S∈𝔉 using set_ultrafilter s(1,3) by auto
qed
```

An ultrafilter is free or principal.

```
corollary ultrafilter_principal_or_free:
  assumes 𝔉{is an ultrafilter on}X
  shows ⋂𝔉 = 0 ∨ (∃x∈X. ⋂𝔉 = {x})
proof(safe)
  assume ¬(∃x∈X. ⋂𝔉 = {x})
  then have r:∀x∈X. ⋂𝔉≠{x} by auto
  {
    fix x assume x:x:X {x}:𝔉
    from this(2) assms have ∀C∈Pow(X). {x} ⊆ C ⟶ C ∈ 𝔉
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have s:{C∈Pow(X). x∈C} ⊆ 𝔉 by auto moreover
    {
      fix q assume q:q∈𝔉 x∉q
      from q(2) s x(1) have X-q:𝔉 by blast
      with q(1) have (X-q)∩q∈𝔉 using assms unfolding IsUltrafilter_def
IsFilter_def by auto
      moreover have (X-q)∩q = 0 by auto
      ultimately have 0:𝔉 by auto
      with assms have False unfolding IsUltrafilter_def IsFilter_def by
auto
    }
    then have 𝔉⊆{C∈Pow(X). x∈C} using assms unfolding IsUltrafilter_def
IsFilter_def by auto
    with s have 𝔉={C∈Pow(X). x∈C} by auto
    then have ⋂𝔉={x} using x(2) by auto
```

```
    with r x(1) have False by auto
  }
  then have ∀x∈X. {x}∉𝔉 by auto
  then have sub:(Cofinite(X))-{0} ⊆ 𝔉 using ultrafilter_contains_cofinite
assms by auto moreover
  from assms have a:X≠0 unfolding IsUltrafilter_def IsFilter_def by auto
  have b:X ∈(Cofinite(X)) using carr_open[of Cofinite(X)]
      union_cocardinal[of nat X] CoCar_is_topology[OF InfCard_nat] un-
folding Cofinite_def by auto
  from a b have X∈(Cofinite(X))-{0} by auto
  with sub have A:⋂𝔉 ⊆ ⋂((Cofinite(X))-{0}) by auto
  {
    fix q assume q:q∈⋂((Cofinite(X))-{0})
    from a b have B:⋂((Cofinite(X))-{0}) ⊆ X by auto
    with q have q∈X by auto
    then have X-(X-{q}) = {q} by auto
    then have X-(X-{q})≈1 using singleton_eqpoll_1 by auto
    then have X-(X-{q}) ≺ nat using eq_lesspoll_trans n_lesspoll_nat[OF
nat_1I] by auto
    then have cof:X-{q}∈(Cofinite(X)) unfolding Cofinite_def CoCardinal_def
by auto
    {
      assume X-{q} = 0
      then have X={q} using `q:X` by auto
      then have ⋂𝔉 ⊆ {q}  using assms unfolding IsUltrafilter_def IsFilter_def
by auto
      with `q∈X` r have ⋂𝔉 = 0 by blast
    } moreover
    {
      assume X-{q} ≠ 0
      with cof have X-{q}∈(Cofinite(X)) -{0} by auto
      then have ⋂((Cofinite(X))-{0}) ⊆ X-{q} by auto
      with q have False by auto
      then have ⋂𝔉 = 0 by auto
    } ultimately
    have ⋂𝔉 = 0 by auto
  }
  then have ⋂𝔉=0 ∨ ⋂((Cofinite(X))-{0}) = 0 by blast
  with A show ⋂𝔉=0 by auto
qed



end
```

91 Ultraproduct construction

```
theory UltraConstruction_ZF imports Ultrafilter_ZF EquivClass1
begin
```

This theory deals with the ultra product construction, internal entities and some basic properties of them.

```
locale ultra =
  fixes 𝔉 and I and X
  assumes ultraFilter:𝔉{is an ultrafilter on}I and nonEmpty:∀i∈I. X(i)≠0

begin
```

We define an equivalence relation

```
definition hyper_rel where
hyper_rel ≡ {⟨f,g⟩∈Pi(I,X)×Pi(I,X). {n∈I. fn = gn}∈𝔉}
```

The relation is reflexive

```
lemma hyper_refl:
  shows refl(Pi(I,X),hyper_rel) unfolding refl_def
proof-
  {
    fix x assume x:x:Pi(I,X)
    have {n∈I. xn = xn} = I by auto moreover
    have I∈𝔉 using ultraFilter unfolding IsUltrafilter_def IsFilter_def
by auto
    ultimately have {n∈I. xn = xn}∈𝔉 by auto
    with x have ⟨x,x⟩∈hyper_rel unfolding hyper_rel_def by auto
  }
  then show ∀x∈Pi(I, X). ⟨x, x⟩ ∈ hyper_rel by auto
qed
```

The relation is symmetric

```
lemma hyper_sym:
  shows sym(hyper_rel) unfolding sym_def
proof(safe)
  fix x y assume ⟨x, y⟩ ∈ hyper_rel
  then have as:x:Pi(I,X) y:Pi(I,X) {n∈I. xn = yn}∈𝔉 unfolding hyper_rel_def
by auto
  {
    fix n assume n∈{n∈I. xn = yn}
    then have n∈I xn=yn by auto
    then have n∈{n∈I. yn = xn} by auto
  } moreover
  {
    fix n assume n∈{n∈I. yn = xn}
    then have n∈I yn=xn by auto
    then have n∈{n∈I. xn = yn} by auto
  } ultimately
  have {n∈I. yn = xn} = {n∈I. xn = yn} by blast
  with as(3) have {n∈I. yn = xn} :𝔉 by auto
  with as(1,2) show ⟨y,x⟩∈hyper_rel unfolding hyper_rel_def by auto
qed
```

The relation is transitive

```
lemma hyper_trans:
  shows trans(hyper_rel) unfolding trans_def
proof(safe)
  fix x y z assume as:⟨x, y⟩ ∈ hyper_rel ⟨y, z⟩ ∈ hyper_rel
  then have A:x:Pi(I,X) z:Pi(I,X) unfolding hyper_rel_def by auto
  {
    fix n assume n∈{n∈I. xn = yn}∩{n∈I. yn = zn}
    then have n∈{n∈I. xn = zn} by auto
  }
  then have sub:{n∈I. xn = yn}∩{n∈I. yn = zn} ⊆ {n∈I. xn = zn} by auto
  from as(1,2) have {n∈I. xn = yn}∩{n∈I. yn = zn}:𝔉 using ultraFilter
    unfolding hyper_rel_def IsUltrafilter_def IsFilter_def by auto
  then have ∀A∈Pow(I). {n∈I. xn = yn}∩{n∈I. yn = zn} ⊆ A ⟶ A∈𝔉
    using ultraFilter unfolding IsUltrafilter_def IsFilter_def by auto
  moreover have {n∈I. xn = zn}∈Pow(I) by auto
  ultimately have ({n∈I. xn = yn}∩{n∈I. yn = zn} ⊆ {n∈I. xn = zn}) ⟶
{n∈I. xn = zn}∈𝔉
    by auto
  with sub have {n∈I. xn = zn}∈𝔉 by auto
  with A show ⟨x, z⟩ ∈ hyper_rel unfolding hyper_rel_def by auto
qed
```

The relation is an equivalence

```
lemma hyper_equiv:
  shows equiv(Pi(I,X), hyper_rel) unfolding equiv_def
  using hyper_refl hyper_sym hyper_trans apply auto
  unfolding hyper_rel_def by auto

definition hyper_set where
hyper_set ≡ Pi(I,X)// hyper_rel

lemma incl_inj:
  fixes Y
  assumes ∀i∈I. X(i) = Y
  defines incl ≡ {⟨x,hyper_rel{ConstantFunction(I,x)}⟩. x∈Y}
  shows incl∈inj(Y,hyper_set) unfolding inj_def
proof(safe)
  {
    fix x assume x:x:Y
    then have ConstantFunction(I,x):I→Y using func1_3_L1 by auto
    then have ConstantFunction(I,x)∈Pi(I,X) unfolding Pi_def Sigma_def
using assms(1) by auto
    then have hyper_rel{ConstantFunction(I,x)}∈hyper_set unfolding hyper_set_def
      quotient_def by auto
  }
  then have ∀x∈Y. hyper_rel{ConstantFunction(I,x)}∈hyper_set by auto
  then show f:incl:Y→hyper_set using ZF_fun_from_total unfolding incl_def
by auto
```

```
  {
    fix w x assume as:w:Y x:Y incl  w = incl  x
    then have e:hyper_rel{ConstantFunction(I,w)} = hyper_rel{ConstantFunction(I,x)}
      using apply_equality[OF _ f] unfolding incl_def by auto
    from as(1,2) have c:ConstantFunction(I,w):I→Y ConstantFunction(I,x):I→Y
      using func1_3_L1 by auto
    then have cc:ConstantFunction(I,w)∈Pi(I,X) ConstantFunction(I,x):Pi(I,X)
unfolding Pi_def Sigma_def using assms(1) by auto
    with e have ⟨ConstantFunction(I,w),ConstantFunction(I,x)⟩∈hyper_rel
using same_image_equiv[of Pi(I,X) hyper_rel]
      hyper_equiv by auto
    then have {n∈I. ConstantFunction(I,w)n = ConstantFunction(I,x)n}:𝔉
      unfolding hyper_rel_def by auto
    then have {n∈I. w = x}:𝔉 using func1_3_L2 by auto
    then show w=x using ultraFilter unfolding IsUltrafilter_def IsFilter_def
by auto
  }
qed

lemma hyper_set_structure:
  assumes ∀i∈I. P(i):X(i)×X(i)→X(i)
  defines Q ≡ {⟨⟨q1,q2⟩,{⟨i,P(i)⟨q1i,q2i⟩⟩. i∈I}⟩. ⟨q1,q2⟩∈Pi(I,X)×Pi(I,X)}
  shows Congruent2(hyper_rel, Q) Q:Pi(I,X)×Pi(I,X) → Pi(I,X)
    ∀i∈I. ∀x∈Pi(I,X). ∀y∈Pi(I,X). Q⟨x,y⟩i = P(i)⟨xi,yi⟩
  unfolding Congruent2_def hyper_rel_def apply safe apply auto
proof-
  {
    fix x y assume xy:x∈Pi(I,X) y∈Pi(I,X)
    {
      fix i assume i:i:I
      with xy have xi:X(i) yi:X(i) using apply_type by auto
      with assms(1) have P(i)⟨xi,yi⟩∈X(i) using apply_type i by auto
    }
    then have {⟨i,P(i)⟨xi,yi⟩⟩. i∈I}:Pi(I,X)
      unfolding Pi_def function_def by auto
  }
  then have ∀x∈Pi(I, X) × Pi(I, X).
       (λ⟨q1,q2⟩. {⟨i, P(i)  ⟨q1  i, q2  i⟩⟩ . i ∈ I})(x) ∈ Pi(I, X) by
auto moreover
  have Q={⟨x, (λ⟨q1,q2⟩. {⟨i, P(i)  ⟨q1  i, q2  i⟩⟩ . i ∈ I})(x)⟩ . x ∈
Pi(I, X) × Pi(I, X)}
    unfolding Q_def by force
  ultimately show qF:Q:Pi(I,X)×Pi(I,X)→Pi(I,X) unfolding Q_def
    using ZF_fun_from_total[of Pi(I,X)×Pi(I,X) λ⟨q1,q2⟩. {⟨i,P(i)⟨q1i,q2i⟩⟩.
i∈I} Pi(I,X)]
    by auto
  then have ∀x∈Pi(I,X). ∀y∈Pi(I,X). Q⟨x,y⟩ = {⟨i,P(i)⟨xi,yi⟩⟩. i∈I}
    using apply_equality unfolding Q_def by auto moreover
  from qF have ∀x∈Pi(I,X). ∀y∈Pi(I,X). Q⟨x,y⟩∈Pi(I,X)
```

```
    using apply_type by auto ultimately
  have R:∀i∈I. ∀x∈Pi(I,X). ∀y∈Pi(I,X). Q⟨x,y⟩i = P(i)⟨xi,yi⟩
    using apply_equality[where A=I and B=X] by auto
  then show ⋀i x y. i ∈ I ⟹ x ∈ Pi(I, X) ⟹ y ∈ Pi(I, X) ⟹ Q
⟨x, y⟩  i = P(i)  ⟨x  i, y  i⟩ by auto
  fix x y z t
  assume as:{n ∈ I . x  n = y  n} ∈ 𝔉
       x ∈ Pi(I,X) y ∈ Pi(I,X) {n ∈ I . z  n = t  n} ∈ 𝔉
       z ∈ Pi(I,X) t ∈ Pi(I,X)
  from qF as(2,5) show Q⟨x,z⟩:Pi(I,X) using apply_type by auto
  from qF as(3,6) show Q⟨y,t⟩:Pi(I,X) using apply_type by auto
  {
    fix n assume n∈{n ∈ I . x  n = y  n}∩{n ∈ I . z  n = t  n}
    then have n∈I x  n = y  n z  n = t  n by auto
    then have P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩ by auto
    with `n∈I` have n∈{n ∈ I . P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩} by auto
  }
  then have {n ∈ I . x  n = y  n}∩{n ∈ I . z  n = t  n} ⊆ {n ∈ I . P(n)⟨xn,zn⟩
= P(n)⟨yn,tn⟩}
    by blast
  moreover from as(1,4) have {n ∈ I . x  n = y  n}∩{n ∈ I . z  n = t
n}:𝔉
    using ultraFilter unfolding IsUltrafilter_def IsFilter_def by auto
  then have ∀A∈Pow(I). {n ∈ I . x  n = y  n}∩{n ∈ I . z  n = t  n} ⊆
A ⟶ A:𝔉
    using ultraFilter unfolding IsUltrafilter_def IsFilter_def by auto
  then have {n ∈ I . P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩}∈Pow(I) ⟶ (({n ∈ I .
x  n = y  n}∩{n ∈ I . z  n = t  n} ⊆ {n ∈ I . P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩})
⟶ {n ∈ I . P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩}:𝔉)
    by auto
  then have ({n ∈ I . x  n = y  n}∩{n ∈ I . z  n = t  n} ⊆ {n ∈ I .
P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩}) ⟶ {n ∈ I . P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩}:𝔉 by
auto
  ultimately have {n ∈ I . P(n)⟨xn,zn⟩ = P(n)⟨yn,tn⟩}:𝔉 by auto
  then show {n ∈ I . Q⟨x,z⟩n = Q⟨y,t⟩n}:𝔉
    using R as(2,3,5,6) by auto
qed
```

91.1 Internal sets

```
definition internal_set where
S:Pi(I,λi. Pow(X(i))) ⟹ internal_set(S) ≡ {hyper_rel{x}. x∈{x∈Pi(I,X).
{n∈I. xn∈Sn}∈𝔉}}

lemma internal_subset:
  assumes S:Pi(I,λi. Pow(X(i)))
  shows internal_set(S) ⊆ hyper_set
proof-
  {
```

```
    fix t assume t∈internal_set(S)
    then have t∈{hyper_rel{x}. x∈{x∈Pi(I,X). {n∈I. xn∈Sn}∈𝔉}}
      unfolding internal_set_def[OF assms].
    then obtain q where q:t=hyper_rel{q} q:Pi(I,X) {n∈I. qn∈Sn}∈𝔉
      by auto
    from q(1,2) have t∈hyper_set unfolding hyper_set_def quotient_def
by auto
  }
  then show internal_set(S) ⊆ hyper_set by auto
qed

lemma internal_sub:
  assumes S1:Pi(I,λi. Pow(X(i)))  S2:Pi(I,λi. Pow(X(i))) {n∈I. S1n ⊆
S2n}∈𝔉
  shows internal_set(S1) ⊆ internal_set(S2)
proof
  fix x assume x∈internal_set(S1)
  then obtain xx where x:xx∈Pi(I,X) x=hyper_rel{xx} {n∈I. xxn∈S1n}∈𝔉
    unfolding internal_set_def[OF assms(1)] by auto
  from x(3) assms(3) have {n∈I. S1n ⊆ S2n}∩{n∈I. xxn∈S1n}∈𝔉 using ultraFilter
    unfolding IsFilter_def IsUltrafilter_def by auto
  moreover have {n∈I. xxn∈S2n}∈Pow(I) by auto
  moreover have {n∈I. S1n ⊆ S2n}∩{n∈I. xxn∈S1n} ⊆ {n∈I. xxn∈S2n} by
auto
  ultimately have {n∈I. xxn∈S2n}∈𝔉 using ultraFilter unfolding IsFilter_def
IsUltrafilter_def
    by auto
  with x(1,2) show x∈internal_set(S2) unfolding internal_set_def[OF assms(2)]
by auto
qed

lemma internal_sub_rev:
  assumes S1:Pi(I,λi. Pow(X(i)))  S2:Pi(I,λi. Pow(X(i)))
  and internal_set(S1) ⊆ internal_set(S2)
  and Pi({i∈I. ¬(S1i⊆S2i)},λj. S1j-S2j) ≠ 0
  and Pi(I,X)≠0
  shows {n∈I. S1n ⊆ S2n}∈𝔉
proof-
  from assms(5) obtain y where y:y∈Pi(I,X) by auto
  from assms(4) obtain x where x:x∈Pi({i∈I. ¬(S1i⊆S2i)},λj. S1j-S2j)
by auto
  let u={⟨i,if S1i⊆S2i then yi else xi⟩. i∈I}
  {
    fix n assume n∈I
    {
      assume S1n ⊆ S2n
      then have (if S1n⊆S2n then yn else xn) = yn by auto
      then have (if S1n⊆S2n then yn else xn)∈X(n) using apply_type `n∈I`
y by auto
```

```
    } moreover
    {
      assume as:¬(S1n ⊆ S2n)
      then have (if S1n⊆S2n then yn else xn) = xn by auto
      moreover have n∈{i∈I. ¬(S1i⊆S2i)} using `n∈I` as by auto
      then have xn∈S1n - S2n using apply_type[OF x, of n] by auto
      ultimately have (if S1n⊆S2n then yn else xn)∈S1n-S2n using apply_type
x `n∈I` x as by auto
      then have (if S1n⊆S2n then yn else xn)∈S1n by auto
      then have (if S1n⊆S2n then yn else xn)∈X(n) using apply_type[OF
assms(1) `n∈I`] by blast
    }
    ultimately have (if S1n⊆S2n then yn else xn)∈X(n) by auto
  }
  then have u:u∈Pi(I,X) unfolding Pi_def function_def by auto
  then have R:⋀n. n∈I ⟹ ¬(S1n ⊆ S2n) ⟹ un=xn using apply_equality
by auto
  {
    assume as:{n∈I. S1n ⊆ S2n}∉𝔉
    with as have I-{n∈I. S1n ⊆ S2n}∈𝔉 using ultraFilter set_ultrafilter[of
_ I 𝔉] by auto
    moreover have {n∈I. ¬(S1n ⊆ S2n)} = I-{n∈I. S1n ⊆ S2n} by auto
    ultimately have F1:{n∈I. ¬(S1n ⊆ S2n)}∈𝔉 by auto
    {
      fix j assume j∈{n∈I. ¬(S1n ⊆ S2n)}
      then have j:j∈I ¬(S1j ⊆ S2j) by auto
      with x have xj ∈ S1j - S2j using apply_type[OF x] by auto
      then have xj∈S1j xj∉S2j by auto
      with R j have uj∈S1j uj∉S2j by auto
    }
    then have F2:{n∈I. ¬(S1n ⊆ S2n)} ⊆ {n∈I. un∈S1n} {n∈I. ¬(S1n ⊆
S2n)} ⊆ {n∈I. un∉S2n} by auto
    from ultraFilter have ⋀C. C⊆I ⟹ {n∈I. ¬(S1n ⊆ S2n)} ⊆ C ⟹ C∈𝔉
      using F1 unfolding IsUltrafilter_def IsFilter_def by auto
    then have {n∈I. un∈S1n}⊆I ⟹ {n∈I. ¬(S1n ⊆ S2n)} ⊆ {n∈I. un∈S1n}
⟹ {n∈I. un∈S1n}∈𝔉
     {n∈I. un∉S2n}⊆I ⟹ {n∈I. ¬(S1n ⊆ S2n)} ⊆ {n∈I. un∉S2n} ⟹ {n∈I.
un∉S2n}∈𝔉 by auto
    with F2 have U:{n∈I. un∈S1n}∈𝔉 {n∈I. un∉S2n}∈𝔉 by auto
    from U(1) u have u1:hyper_rel{u}∈internal_set(S1) unfolding internal_set_def[OF
assms(1)]
      by auto
    have I-{n∈I. un∈S2n} = {n∈I. un∉S2n} by auto
    with U(2) have I-{n∈I. un∈S2n}∈𝔉 by auto
    then have I-(I-{n∈I. un∈S2n})∉𝔉 using ultraFilter only_set_or_opposite[of
I-{n∈I. un∈S2n} I]
      unfolding IsUltrafilter_def by auto
    moreover have I-(I-{n∈I. un∈S2n}) = {n∈I. un∈S2n} by auto
    ultimately have M:{n∈I. un∈S2n}∉𝔉 by auto
```

```
    {
      fix q assume q∈Pi(I,X) hyper_rel{q} = hyper_rel{u}
      then have ⟨q,u⟩∈hyper_rel using same_image_equiv[OF hyper_equiv]
u by auto
      then have A:{n∈I. qn = un}∈𝔉 unfolding hyper_rel_def by auto
      {
        assume {n∈I. qn∈S2n}∈𝔉
        with A have AA:{n∈I. qn = un}∩{n∈I. qn∈S2n}∈𝔉 using ultraFilter
unfolding IsUltrafilter_def
          using is_filter_def_split(4) by auto
        moreover have B:{n∈I. qn = un}∩{n∈I. qn∈S2n} ⊆ {n∈I. un∈S2n}
by auto
        from AA ultraFilter have ⋀C. C⊆ I ⟹ {n∈I. qn = un}∩{n∈I.
qn∈S2n} ⊆ C ⟹ C∈𝔉
          unfolding IsUltrafilter_def using is_filter_def_split(5) by
auto
        then have {n∈I. un∈S2n}⊆I ⟹{n∈I. qn = un}∩{n∈I. qn∈S2n} ⊆
{n∈I. un∈S2n} ⟹ {n∈I. un∈S2n}∈𝔉
          by auto
        then have {n∈I. qn = un}∩{n∈I. qn∈S2n} ⊆ {n∈I. un∈S2n} ⟹
{n∈I. un∈S2n}∈𝔉 by auto
        with B have {n∈I. un∈S2n}∈𝔉 by auto
        with M have False by auto
      }
      then have {n∈I. qn∈S2n}∉𝔉 by auto
    }
    then have hyper_rel{u}∉internal_set(S2) unfolding internal_set_def[OF
assms(2)] by auto
    with u1 assms(3) have False by auto
  }
  then show thesis by auto
qed

corollary internal_eq:
  assumes S1:Pi(I,λi. Pow(X(i)))  S2:Pi(I,λi. Pow(X(i))) {n∈I. S1n =
S2n}∈𝔉
  shows internal_set(S1) = internal_set(S2)
proof
  have {n∈I. S1n = S2n} ⊆ {n∈I. S1n ⊆ S2n} {n∈I. S1n = S2n} ⊆ {n∈I.
S2n ⊆ S1n} by auto
  moreover have {n∈I. S1n ⊆ S2n}:Pow(I) {n∈I. S2n ⊆ S1n}:Pow(I) by
auto
  moreover note assms(3) ultimately have A:{n∈I. S1n ⊆ S2n}:𝔉 {n∈I.
S2n ⊆ S1n}:𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
  from A(1) show internal_set(S1) ⊆ internal_set(S2) using internal_sub[OF
assms(1,2)] by auto
  from A(2) show internal_set(S2) ⊆ internal_set(S1) using internal_sub[OF
assms(2,1)] by auto
```

```
qed

lemma internal_total_set:
  shows internal_set({⟨i,X(i)⟩. i∈I}) = hyper_set
proof
  have f:{⟨i,X(i)⟩. i∈I}:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def
by auto
  {
    fix t assume t∈internal_set({⟨i,X(i)⟩. i∈I})
    then have t∈{hyper_rel{x}. x∈{x∈Pi(I,X). {n∈I. xn∈{⟨i,X(i)⟩. i∈I}n}∈𝔉}}
      unfolding internal_set_def[OF f].
    then obtain q where q:t=hyper_rel{q} q:Pi(I,X) {n∈I. qn∈{⟨i,X(i)⟩.
i∈I}n}∈𝔉
      by auto
    from q(1,2) have t∈hyper_set unfolding hyper_set_def quotient_def
by auto
  }
  then show internal_set({⟨i,X(i)⟩. i∈I}) ⊆ hyper_set by auto
  {
    fix t assume t∈hyper_set
    then obtain q where q:t=hyper_rel{q} q:Pi(I,X) unfolding hyper_set_def
        quotient_def by auto
    from f have R:∀n∈I. {⟨i,X(i)⟩. i∈I}n = X(n) using apply_equality
by auto
    from q(2) have ∀n∈I. qn∈X(n) using apply_type by auto
    then have {n∈I. qn∈X(n)} = I by auto
    then have {n∈I. qn∈X(n)}∈𝔉 using ultraFilter unfolding IsUltrafilter_def
IsFilter_def
      by auto
    with R have {n∈I. qn∈{⟨i,X(i)⟩. i∈I}n}∈𝔉 by auto
    with q(2) have q∈{x∈Pi(I,X). {n∈I. xn∈{⟨i,X(i)⟩. i∈I}n}∈𝔉} by auto
    with q(1) have t∈{hyper_rel{x}. x∈{x∈Pi(I,X). {n∈I. xn∈{⟨i,X(i)⟩.
i∈I}n}∈𝔉}} by auto
    then have t:internal_set({⟨i,X(i)⟩. i∈I}) unfolding internal_set_def[OF
f].
  }
  then show hyper_set ⊆ internal_set({⟨i,X(i)⟩. i∈I}) by auto
qed

definition isInternal where
isInternal(Y) ≡ ∃S∈Pi(I,λi. Pow(X(i))). Y=internal_set(S)

corollary total_inter:
  shows isInternal(hyper_set)
proof-
  have {⟨i,X(i)⟩. i∈I}:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def
by auto
  then show thesis unfolding isInternal_def using internal_total_set[THEN
sym] by auto
```

```
qed

lemma internal_inter:
  assumes isInternal(A) isInternal(B)
  shows isInternal(A∩B)
proof-
  from assms obtain SA SB where s:SA:Pi(I,λi. Pow(X(i))) SB:Pi(I,λi.
Pow(X(i)))
    A=internal_set(SA) B=internal_set(SB)
    unfolding isInternal_def by auto
  let S={⟨n,(SAn)∩(SBn)⟩. n∈I}
  from s(1,2) have ∀n∈I. (SAn)∈Pow(X(n)) ∀n∈I. (SBn)∈Pow(X(n)) using
apply_type[of _ I λi. Pow(X(i))]
    by auto
  then have ∀n∈I. (SAn)∩(SBn)∈Pow(X(n)) by auto
  then have f:S:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def by
auto
  {
    fix t assume t:t∈internal_set(S)
    then obtain x where x:t=hyper_rel{x} x∈Pi(I,X) {n∈I. xn∈Sn}∈𝔉
      unfolding internal_set_def[OF f] by auto
    from x(3) have {n∈I. xn∈(SAn)∩(SBn)}∈𝔉
      using apply_equality[OF _ f] by auto
    then have R:∀A∈Pow(I). {n∈I. xn∈(SAn)∩(SBn)} ⊆ A ⟶ A∈𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    moreover have {n∈I. xn∈SAn}∈Pow(I) by auto
    ultimately have {n∈I. xn∈(SAn)∩(SBn)} ⊆ {n∈I. xn∈SAn} ⟶ {n∈I.
xn∈SAn}∈𝔉 by auto
    moreover have {n∈I. xn∈(SAn)∩(SBn)} ⊆ {n∈I. xn∈SAn} by auto
    ultimately have {n∈I. xn∈SAn}∈𝔉 by auto
    with x(1,2) have A:t∈internal_set(SA) unfolding internal_set_def[OF
s(1)] by auto
    have {n∈I. xn∈SBn}∈Pow(I) by auto
    with R have {n∈I. xn∈(SAn)∩(SBn)} ⊆ {n∈I. xn∈SBn} ⟶ {n∈I. xn∈SBn}∈𝔉
by auto
    moreover have {n∈I. xn∈(SAn)∩(SBn)} ⊆ {n∈I. xn∈SBn} by auto
    ultimately have {n∈I. xn∈SBn}∈𝔉 by auto
    with x(1,2) have t∈internal_set(SB) unfolding internal_set_def[OF
s(2)] by auto
    with A have t∈A∩B using s(3,4) by auto
  }
  then have internal_set(S) ⊆ A∩B by auto moreover
  {
    fix t assume t:t∈A∩B
    with s(3,4) obtain ta tb where t:t=hyper_rel{ta} ta∈Pi(I,X) {n∈I.
tan∈SAn}∈𝔉
      t=hyper_rel{tb} tb∈Pi(I,X) {n∈I. tbn∈SBn}∈𝔉 unfolding internal_set_def[OF
s(1)]
      internal_set_def[OF s(2)] by blast
```

```
    from t(1,4) have hyper_rel{ta}=hyper_rel{tb} by auto
    then have ⟨ta,tb⟩∈hyper_rel using same_image_equiv[OF hyper_equiv]
t(5) by auto
    then have {n∈I. tan=tbn}:𝔉 unfolding hyper_rel_def by auto
    with t(3) have {n∈I. tan=tbn}∩{n∈I. tan∈SAn}:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    with t(6) have {n∈I. tan=tbn}∩{n∈I. tan∈SAn}∩{n∈I. tbn∈SBn}:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    then have ∀A∈Pow(I). {n∈I. tan=tbn}∩{n∈I. tan∈SAn}∩{n∈I. tbn∈SBn}⊆A
⟶ A:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    moreover have {n∈I. tan∈(SAn)∩SBn} :Pow(I) by auto
    ultimately have {n∈I. tan=tbn}∩{n∈I. tan∈SAn}∩{n∈I. tbn∈SBn}⊆{n∈I.
tan∈(SAn)∩SBn} ⟶ {n∈I. tan∈(SAn)∩SBn}:𝔉
      by auto
    then have {n∈I. tan∈(SAn)∩SBn}:𝔉 by auto
    then have {n∈I. tan∈{⟨n, SA  n ∩ SB  n⟩ . n ∈ I}n}:𝔉
      using apply_equality[OF _ f] by auto
    with t(1,2) have t∈internal_set(S) unfolding internal_set_def[OF
f] by auto
  }
  then have A∩B ⊆ internal_set(S) by auto
  ultimately have A∩B = internal_set(S) by auto
  then show thesis unfolding isInternal_def using f by auto
qed
```

The complement of an internal set is internal

```
lemma complement_internal:
  assumes isInternal(A)
  shows isInternal(hyper_set-A)
proof-
  from assms obtain SA where s:SA:Pi(I,λi. Pow(X(i)))
    A=internal_set(SA) unfolding isInternal_def by auto
  let S={⟨n,X(n)-(SAn)⟩. n∈I}
  have ∀n∈I. X(n)-(SAn)∈Pow(X(n)) by auto
  then have f:S:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def by
auto
  {
    fix t assume t:t∈internal_set(S)
    then obtain x where x:t=hyper_rel{x} x∈Pi(I,X) {n∈I. xn∈Sn}∈𝔉
      unfolding internal_set_def[OF f] by auto
    {
      assume t∈internal_set(SA)
      then obtain y where y:t=hyper_rel{y} y∈Pi(I,X) {n∈I. yn∈SAn}∈𝔉
        unfolding internal_set_def[OF s(1)] by auto
      from x(1) y(1) have ⟨x,y⟩∈hyper_rel using y(2) same_image_equiv
          hyper_equiv by auto
      then have {n∈I. xn = yn}∈𝔉 unfolding hyper_rel_def by auto
      with y(3) have {n∈I. yn∈SAn}∩{n∈I. xn = yn}∈𝔉
```

```
        using ultraFilter unfolding IsFilter_def IsUltrafilter_def by
auto
      then have ⋀A. A∈Pow(I) ⟹ {n∈I. yn∈SAn}∩{n∈I. xn = yn} ⊆ A ⟹
A:𝔉
        using ultraFilter unfolding IsFilter_def IsUltrafilter_def by
auto
      then have {n∈I. xn∈SAn}∈Pow(I) ⟹ {n∈I. yn∈SAn}∩{n∈I. xn = yn}
⊆ {n∈I. xn∈SAn} ⟹ {n∈I. xn∈SAn}:𝔉 by auto
      moreover have {n∈I. yn∈SAn}∩{n∈I. xn = yn} ⊆ {n∈I. xn∈SAn} by
auto
      ultimately have {n∈I. xn∈SAn}: 𝔉 by auto
      with x(3) have {n∈I. xn∈SAn}∩{n∈I. xn∈X(n)-(SAn)}∈𝔉 using ultraFilter
        apply_equality[OF _ f]
        unfolding IsUltrafilter_def IsFilter_def by auto moreover
      {
        fix n assume n:n∈{n∈I. xn∈SAn}∩{n∈I. xn∈X(n)-(SAn)}
        then have False by auto
      }
      then have {n∈I. xn∈SAn}∩{n∈I. xn∈X(n)-(SAn)} = 0 by auto
      ultimately have 0∈𝔉 by auto
      then have False using ultraFilter
        unfolding IsUltrafilter_def IsFilter_def by auto
    }
    then have t∉internal_set(SA) by auto moreover
    have t∈hyper_set using internal_subset[OF f] t by auto
    ultimately have t∈hyper_set-A using s(2) by auto
  }
  then have internal_set(S) ⊆ hyper_set-A by auto moreover
  {
    fix t assume t∈hyper_set-A
    then have t:t∈hyper_set t∉A by auto
    from t(1) obtain x where x:t=hyper_rel{x} x:Pi(I,X) unfolding hyper_set_def
      quotient_def by auto
    with t(2) s(2) have {n∈I. xn∈SAn}∉𝔉 unfolding internal_set_def[OF
s(1)] by auto
    then have I-{n∈I. xn∈SAn}∈𝔉 using set_ultrafilter[OF _ ultraFilter]
      by auto
    then have {n∈I. xn∈X(n)-SAn}∈Pow(I) ⟹ I-{n∈I. xn∈SAn} ⊆ {n∈I.
xn∈X(n)-SAn} ⟶ {n∈I. xn∈X(n)-SAn}:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
    {
      fix q assume q∈I-{n∈I. xn∈SAn}
      then have q: I xq∉SAq by auto
      with x(2) have q∈I xq:X(q) xq∉SAq using apply_type by auto
      then have q∈{n∈I. xn∈X(n)-SAn} by auto
    }
    then have I-{n∈I. xn∈SAn} ⊆ {n∈I. xn∈X(n)-SAn} by auto ultimately
    have {n∈I. xn∈X(n)-SAn}∈𝔉 by auto
```

```
    then have {n∈I. xn∈Sn}∈𝔉 using apply_equality[OF _ f] by auto
    with x(1,2) have t∈internal_set(S) unfolding internal_set_def[OF
f] by auto
  }
  ultimately have hyper_set-A = internal_set(S) by auto
  then show thesis unfolding isInternal_def using f by auto
qed

lemma finite_internal:
  assumes A∈FinPow(hyper_set)
  shows isInternal(A) unfolding isInternal_def
proof(rule FinPow_induct[OF _ _ assms])
  let S0=I×{0}
  have s:S0:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def by auto
  {
    fix t assume t∈internal_set(S0)
    then obtain x where x:t=hyper_rel{x} x:Pi(I,X) {n∈I. xn∈S0n}∈𝔉
      unfolding internal_set_def[OF s] by auto
    from x(3) have {n∈I. xn∈0}∈𝔉 using apply_equality[OF _ s] by auto
    then have 0∈𝔉 by auto
    then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
  }
  then have 0 = internal_set(S0) by auto
  with s show ∃S∈∏i∈I. Pow(X(i)). 0 = internal_set(S) by auto
  {
    fix B assume b:B∈FinPow(hyper_set) ∃S∈∏i∈I. Pow(X(i)). B = internal_set(S)
    from b(2) obtain S where S:S:(∏i∈I. Pow(X(i))) B=internal_set(S)
by auto

    {
      fix t assume t∈hyper_set
      then obtain x where x:t=hyper_rel{x} x:Pi(I,X) unfolding hyper_set_def
        quotient_def by auto
      let S={⟨i,Si∪{xi}⟩. i∈I}
      have f:S:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def apply
auto
        using apply_type[OF x(2)] apply_type[OF S(1)] by auto
      {
        fix q assume q:q∈B∪{t}
        {
          assume q∈B
          with S(2) obtain y where y:q=hyper_rel{y} y:Pi(I,X) {n∈I. yn∈Sn}:𝔉
            unfolding internal_set_def[OF S(1)] by auto
          have {n∈I. yn∈Sn} ⊆ {n∈I. yn∈Sn∪{xn}} by auto
          moreover from y(3) have ⋀Q. Q ⊆ I ⟹ {n∈I. yn∈Sn} ⊆ Q ⟹
Q:𝔉
            using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
```

```
          from this[of {n∈I. yn∈Sn∪{xn}}]
          have {n∈I. yn∈Sn} ⊆ {n∈I. yn∈Sn∪{xn}} ⟹ {n∈I. yn∈Sn∪{xn}}:𝔉
by auto
          ultimately have {n∈I. yn∈Sn∪{xn}}:𝔉 by auto
          then have {n∈I. yn∈Sn}:𝔉 using apply_equality[OF _ f] by auto
          with y(1,2) have q∈internal_set(S) unfolding internal_set_def[OF
f] by auto
        } moreover
        {
          assume q∉B
          with q have q=t by auto
          with x have xx:q=hyper_rel{x} x:Pi(I,X) by auto
          have {n∈I. xn∈{xn}} = I by auto
          then have {n∈I. xn∈{xn}}∈𝔉 using ultraFilter unfolding IsFilter_def
            IsUltrafilter_def by auto
          then have ⋀Q. Q ⊆ I ⟹ {n∈I. xn∈{xn}} ⊆ Q ⟹ Q:𝔉
            using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
          from this[of {n:I. xn:Sn}] have
            {n∈I. xn∈{xn}} ⊆ {n:I. xn:Sn} ⟹ {n:I. xn:Sn}:𝔉 by auto
          moreover
          have {n∈I. xn∈{xn}} ⊆ {n:I. xn:Sn} using apply_equality[OF
_ f] by auto
          ultimately have  {n:I. xn:Sn}:𝔉 by auto
          with xx have q∈internal_set(S) unfolding internal_set_def[OF
f] by auto
        } ultimately
        have q:internal_set(S) by auto
      }
      then have B∪{t} ⊆ internal_set(S) by auto moreover
      {
        fix q assume q:q∈internal_set(S) q≠t
        then obtain y where y:q=hyper_rel{y} y:Pi(I,X) {n:I. yn:Sn}:𝔉
          unfolding internal_set_def[OF f] by auto
        {
          assume {n∈I. yn∈Sn}∉𝔉
          then have I-{n∈I. yn∈Sn}∈𝔉 using ultraFilter
            set_ultrafilter[of _ I, OF _ ultraFilter] by auto
          with y(3) have (I-{n∈I. yn∈Sn})∩{n:I. yn:Sn}:𝔉
            using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
          then have ⋀B. B:Pow(I) ⟹ (I-{n∈I. yn∈Sn})∩{n:I. yn:Sn} ⊆
B ⟹ B:𝔉
            using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
          from this[of {n:I. yn=xn}] have (I-{n∈I. yn∈Sn})∩{n:I. yn:Sn}
⊆ {n:I. yn=xn} ⟹ {n:I. yn=xn}:𝔉
            by auto moreover
          {
```

```
          fix h assume h:(I-{n∈I. yn∈Sn})∩{n:I. yn:Sn}
          then have h∈I yh∉Sh yh∈Sh by auto
          then have h∈I yh = xh using apply_equality[OF _ f] by auto
          then have h:{n:I. yn=xn} by auto
        }
        then have (I-{n∈I. yn∈Sn})∩{n:I. yn:Sn} ⊆ {n:I. yn=xn} by
auto
        ultimately have {n:I. yn=xn}:𝔉 by auto
        with x(2) y(2) have ⟨y,x⟩:hyper_rel unfolding hyper_rel_def
by auto
        then have q=t using x(1) y(1) equiv_class_eq[OF hyper_equiv]
by auto
        with q(2) have False by auto
      }
      then have {n∈I. yn∈Sn}∈𝔉 by auto
      with y(1,2) have q∈B using S(2) unfolding internal_set_def[OF
S(1)] by auto
    }
    then have internal_set(S) ⊆ B∪{t} by auto
    ultimately have B∪{t} = internal_set(S) by auto
    with f have ∃S∈∏i∈I. Pow(X(i)). B ∪ {t} = internal_set(S) by
auto
  }
  then have ∀t∈hyper_set. ∃S∈∏i∈I. Pow(X(i)). B ∪ {t} = internal_set(S)
by auto
 }
 then show ∀A∈FinPow(hyper_set).
     (∃S∈∏i∈I. Pow(X(i)). A = internal_set(S)) ⟶
     (∀t∈hyper_set. ∃S∈∏i∈I. Pow(X(i)). A ∪ {t} = internal_set(S))
by auto
qed

lemma internal_union:
 assumes isInternal(A) isInternal(B)
 shows isInternal(A∪B)
proof-
 from assms obtain SA SB where s:SA:Pi(I,λi. Pow(X(i))) SB:Pi(I,λi.
Pow(X(i)))
   A=internal_set(SA) B=internal_set(SB)
   unfolding isInternal_def by auto
 let S={⟨n,(SAn)∪(SBn)⟩. n∈I}
 from s(1,2) have ∀n∈I. (SAn)∈Pow(X(n)) ∀n∈I. (SBn)∈Pow(X(n)) using
apply_type[of _ I λi. Pow(X(i))]
   by auto
 then have ∀n∈I. (SAn)∪(SBn)∈Pow(X(n)) by auto
 then have f:S:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def by
auto
 {
   fix t assume t:t∈internal_set(S)
```

```
    then obtain x where x:t=hyper_rel{x} x∈Pi(I,X) {n∈I. xn∈Sn}∈𝔉
      unfolding internal_set_def[OF f] by auto
    from x(3) have U:{n∈I. xn∈(SAn)∪(SBn)}∈𝔉
      using apply_equality[OF _ f] by auto
    {
      assume t∉A
      with x(1,2) s(3) have {n∈I. xn∈(SAn)}∉𝔉 unfolding internal_set_def[OF
s(1)]
        by auto
      then have I-{n∈I. xn∈(SAn)}∈𝔉 using ultraFilter
        set_ultrafilter[of {n∈I. xn∈(SAn)}] by auto moreover
      have {n∈I. xn∉(SAn)}∈Pow(I) by auto moreover
      {
        fix n assume n∈I-{n∈I. xn∈(SAn)}
        then have n∈{n∈I. xn∉(SAn)} by auto
      }
      then have I-{n∈I. xn∈(SAn)} ⊆ {n∈I. xn∉(SAn)} by auto
      ultimately have {n∈I. xn∉(SAn)}:𝔉 using ultraFilter unfolding IsFilter_def
        IsUltrafilter_def by auto
      with U have {n∈I. xn∈(SAn)∪(SBn)}∩{n∈I. xn∉(SAn)}:𝔉
        using ultraFilter unfolding IsFilter_def IsUltrafilter_def by
auto moreover
      have {n:I. xn∈SBn}∈Pow(I) by auto moreover
      {
        fix n assume n∈{n∈I. xn∈(SAn)∪(SBn)}∩{n∈I. xn∉(SAn)}
        then have n∈I xn∈SBn by auto
        then have n:{n:I. xn∈SBn} by auto
      }
      then have {n∈I. xn∈(SAn)∪(SBn)}∩{n∈I. xn∉(SAn)} ⊆ {n:I. xn∈SBn}
by auto
      ultimately have {n:I. xn∈SBn}∈𝔉 using ultraFilter unfolding IsFilter_def
        IsUltrafilter_def by auto
      with x(1,2) have t∈B using s(4) unfolding internal_set_def[OF s(2)]
by auto
    }
    then have t∈A∪B by auto
  }
  then have internal_set(S) ⊆ A∪B by auto moreover
  {
    fix t assume t∈A
    then obtain x where x:t=hyper_rel{x} x∈Pi(I,X) {n∈I. xn∈SAn}∈𝔉
      using s(3) unfolding internal_set_def[OF s(1)] by auto
    have {n∈I. xn∈SAn} ⊆ {n∈I. xn∈(SAn)∪(SBn)} by auto moreover
    have {n∈I. xn∈(SAn)∪(SBn)}∈Pow(I) by auto moreover note x(3)
    ultimately have {n∈I. xn∈(SAn)∪(SBn)}:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have {n∈I. xn∈Sn}:𝔉 using apply_equality[OF _ f] by auto
    with x(1,2) have t∈internal_set(S) unfolding internal_set_def[OF
f] by auto
```

```
  }
  then have A ⊆ internal_set(S) by auto moreover
  {
    fix t assume t∈B
    then obtain x where x:t=hyper_rel{x} x∈Pi(I,X) {n∈I. xn∈SBn}∈𝔉
      using s(4) unfolding internal_set_def[OF s(2)] by auto
    have {n∈I. xn∈SBn} ⊆ {n∈I. xn∈(SAn)∪(SBn)} by auto moreover
    have {n∈I. xn∈(SAn)∪(SBn)}∈Pow(I) by auto moreover note x(3)
    ultimately have {n∈I. xn∈(SAn)∪(SBn)}:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have {n∈I. xn∈Sn}:𝔉 using apply_equality[OF _ f] by auto
    with x(1,2) have t∈internal_set(S) unfolding internal_set_def[OF
f] by auto
  }
  then have B ⊆ internal_set(S) by auto ultimately
  have A ∪ B = internal_set(S) by auto
  then show thesis unfolding isInternal_def using f by auto
qed

definition elevatedProp (^_^ 90) where
^P^ ≡ λt. ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧ hyper_rel{y}=t

lemma internal_subset_internal:
  assumes isInternal(A)
  shows isInternal({x∈A. (^P^)(x)})
proof-
  from assms obtain S where s:S:Pi(I,λi. Pow(X(i)))
    A=internal_set(S) unfolding isInternal_def by auto
  let S={⟨i,{x∈Si. P(x)}⟩. i∈I}
  {
    fix i assume i:i∈I
    then have {x∈Si. P(x)} ⊆ Si by auto
    with s(1) have {x∈Si. P(x)} ⊆ X(i) using apply_type[OF _ i, of S
λi. Pow(X(i))] by auto
  }
  then have ss:S:Pi(I,λi. Pow(X(i))) unfolding Pi_def function_def by
auto
  {
    fix t assume t∈{x∈A. ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧ hyper_rel{y} =
x}
    then obtain y where t:t∈A y∈Pi(I,X) {i∈I. P(yi)}∈𝔉 hyper_rel{y}
= t by auto
    from t(1,4) have hyper_rel{y}∈internal_set(S) using s(2) by auto
    then obtain z where z:hyper_rel{y} = hyper_rel{z} z∈Pi(I,X) {i∈I.
zi∈Si}∈𝔉
      unfolding internal_set_def[OF s(1)] by auto
    from z(1) have ⟨y,z⟩∈hyper_rel using same_image_equiv[OF hyper_equiv]
z(2) by auto
    then have {i∈I. yi = zi}∈𝔉 unfolding hyper_rel_def by auto
```

```
    with z(3) have {i∈I. yi = zi}∩{i∈I. zi∈Si}∈𝔉 using ultraFilter
      unfolding IsUltrafilter_def IsFilter_def by auto
    moreover have {i∈I. yi = zi}∩{i∈I. zi∈Si} = {i∈I. yi = zi ∧ yi∈Si}
by auto
    ultimately have {i∈I. yi = zi ∧ yi∈Si}∈𝔉 by auto moreover
    have {i∈I. yi = zi ∧ yi∈Si} ⊆ {i∈I. yi∈Si} by auto moreover
    have ⋀C. C⊆I ⟹ {i∈I. yi = zi ∧ yi∈Si} ⊆ C ⟹ {i∈I. yi = zi ∧
yi∈Si}∈𝔉 ⟹ C∈𝔉
      using ultraFilter unfolding IsUltrafilter_def IsFilter_def by auto
    then have {i∈I. yi∈Si} ⊆ I ⟹ {i∈I. yi = zi ∧ yi∈Si} ⊆ {i∈I. yi∈Si}
⟹ {i∈I. yi = zi ∧ yi∈Si}∈𝔉 ⟹ {i∈I. yi∈Si}∈𝔉 by auto
    ultimately have {i∈I. yi∈Si}∈𝔉 by auto
    with t(3) have {i∈I. yi∈Si}∩{i∈I. P(yi)}∈𝔉 using ultraFilter un-
folding IsUltrafilter_def
      IsFilter_def by auto moreover
    have {i∈I. yi∈Si}∩{i∈I. P(yi)} = {i∈I. yi∈{x∈Si. P(x)}} by auto
    ultimately have {i∈I. yi∈{x∈Si. P(x)}}∈𝔉 by auto
    then have {i∈I. yi∈Si}∈𝔉 using apply_equality[OF _ ss] by auto
    with t(2,4) have t∈internal_set(S) unfolding internal_set_def[OF
ss] by auto
  }
  then have {x∈A. ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧ hyper_rel{y} = x} ⊆
internal_set(S) by auto moreover
  {
    fix t assume t∈internal_set(S)
    then obtain y where t:t=hyper_rel{y} y∈Pi(I,X) {i∈I. yi∈Si}∈𝔉
      unfolding internal_set_def[OF ss] by auto
    from t(3) have {i∈I. yi∈{x∈Si. P(x)}}∈𝔉 using apply_equality ss
by auto
    moreover have {i∈I. yi∈{x∈Si. P(x)}} ⊆ {i∈I. yi∈Si} {i∈I. yi∈{x∈Si.
P(x)}} ⊆ {i∈I. P(yi)} by auto
    moreover have ⋀C. C⊆I ⟹ {i∈I. yi∈{x∈Si. P(x)}} ⊆ C ⟹ {i∈I.
yi∈{x∈Si. P(x)}}∈𝔉 ⟹ C∈𝔉
      using ultraFilter unfolding IsUltrafilter_def IsFilter_def by auto
    then have {i∈I. yi∈Si}⊆I ⟹ {i∈I. yi∈{x∈Si. P(x)}} ⊆ {i∈I. yi∈Si}
⟹ {i∈I. yi∈{x∈Si. P(x)}}∈𝔉 ⟹ {i∈I. yi∈Si}∈𝔉
      {i∈I. P(yi)}⊆I ⟹ {i∈I. yi∈{x∈Si. P(x)}} ⊆ {i∈I. P(yi)} ⟹ {i∈I.
yi∈{x∈Si. P(x)}}∈𝔉 ⟹ {i∈I. P(yi)}∈𝔉 by auto
    ultimately have {i∈I. yi∈Si}∈𝔉 {i∈I. P(yi)}∈𝔉 by auto
    with t(1,2) have t∈internal_set(S) ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧
hyper_rel{y} = t
      unfolding internal_set_def[OF s(1)] by auto
    with s(2) have t∈{x∈A. ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧ hyper_rel{y}
= x} by auto
  }
  then have internal_set(S) ⊆ {x∈A. ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧ hyper_rel{y}
= x} by auto
  ultimately have {x∈A. ∃y∈Pi(I,X). {i∈I. P(yi)}∈𝔉 ∧ hyper_rel{y} =
x} = internal_set(S) by auto
```

```
  then show thesis unfolding isInternal_def using ss unfolding elevatedProp_def
by auto
qed


definition internal_rel where
S:Pi(I,λi. Pow(X(i)×X(i))) ⟹ internal_rel(S) ≡ {⟨hyper_rel{x},hyper_rel{y}⟩.
⟨x,y⟩∈{⟨p,q⟩∈Pi(I,X)×Pi(I,X). {n∈I. ⟨pn,qn⟩∈Sn}∈𝔉}}

lemma internal_rel_subset:
  assumes S:Pi(I,λi. Pow(X(i)×X(i)))
  shows internal_rel(S) ⊆ hyper_set×hyper_set
proof-
  {
    fix t assume t∈internal_rel(S)
    then have t∈{⟨hyper_rel{x},hyper_rel{y}⟩. ⟨x,y⟩∈{⟨p,q⟩∈Pi(I,X)×Pi(I,X).
{n∈I. ⟨pn,qn⟩∈Sn}∈𝔉}}
      unfolding internal_rel_def[OF assms].
    then obtain q p where q:t=⟨hyper_rel{q},hyper_rel{p}⟩ q:Pi(I,X) p:Pi(I,X)
{n∈I. ⟨qn,pn⟩∈Sn}∈𝔉
      by auto
    from q(1-3) have t∈hyper_set×hyper_set unfolding hyper_set_def quotient_def
by auto
  }
  then show internal_rel(S) ⊆ hyper_set×hyper_set by auto
qed

lemma converse_internal:
  assumes S:Pi(I,λi. Pow(X(i)×X(i)))
  shows converse(internal_rel(S)) = internal_rel({⟨i,converse(Si)⟩. i∈I})
proof
  have ⋀i. i∈I ⟹ (Si) ⊆ X(i)×X(i) using apply_type[OF assms] by auto
  then have ⋀i. i∈I ⟹ converse(Si) ⊆ X(i)×X(i) unfolding converse_def
by auto
  then have A:{⟨i,converse(Si)⟩. i∈I}:Pi(I,λi. Pow(X(i)×X(i))) unfold-
ing Pi_def function_def by auto
  {
    fix x assume x∈converse(internal_rel(S))
    then obtain y z where x:x=⟨y,z⟩ ⟨z,y⟩∈internal_rel(S) using converse_iff
by auto
    from x(2) obtain zz yy where q:zz:Pi(I,X) yy:Pi(I,X) z=hyper_rel{zz}
y=hyper_rel{yy}
      {n ∈ I . ⟨zz  n, yy  n⟩ ∈ S  n} ∈ 𝔉
      unfolding internal_rel_def[OF assms] by auto
    from q(5) have {n ∈ I . ⟨yy  n, zz  n⟩ ∈ converse(S  n)}:𝔉 using
converse_iff by auto
    then have {n ∈ I . ⟨yy  n, zz  n⟩ ∈ {⟨i,converse(Si)⟩. i∈I}n}:𝔉 us-
ing apply_equality[OF _ A] by auto
    with q(1-4) x(1) have x∈internal_rel({⟨i,converse(Si)⟩. i∈I}) un-
```

```
folding internal_rel_def[OF A] by auto
  }
  then show converse(internal_rel(S)) ⊆ internal_rel({⟨i, converse(S
i)⟩ . i ∈ I}) by auto
  {
    fix x assume x∈internal_rel({⟨i,converse(Si)⟩. i∈I})
    then obtain zz yy where q:zz:Pi(I,X) yy:Pi(I,X) x=⟨hyper_rel{zz},hyper_rel{yy}⟩

      {n ∈ I . ⟨zz  n, yy  n⟩ ∈ {⟨i,converse(Si)⟩. i∈I}  n} ∈ 𝔉
      unfolding internal_rel_def[OF A] by auto
    from q(4) have {n ∈ I . ⟨zz  n, yy  n⟩ ∈ converse(S  n)} ∈ 𝔉  us-
ing apply_equality[OF _ A] by auto
    then have {n ∈ I . ⟨yy  n, zz  n⟩ ∈ (S  n)} ∈ 𝔉 using converse_iff
by auto
    then have ⟨hyper_rel{yy},hyper_rel{zz}⟩∈internal_rel(S) unfolding
internal_rel_def[OF assms]
      using q(1,2) by auto
    then have x:converse(internal_rel(S)) using q(3) converse_iff by
auto
  }
  then show internal_rel({⟨i,converse(Si)⟩. i∈I}) ⊆ converse(internal_rel(S))
by auto
qed


lemma internal_rel_total:
  shows internal_rel({⟨i,X(i)×X(i)⟩. i∈I}) = hyper_set×hyper_set
proof
  have f:{⟨i,X(i)×X(i)⟩. i∈I}:Pi(I,λi. Pow(X(i)×X(i))) unfolding Pi_def
function_def by auto
  then show internal_rel({⟨i,X(i)×X(i)⟩. i∈I}) ⊆ hyper_set×hyper_set
using internal_rel_subset by auto
  {
    fix m assume m∈hyper_set×hyper_set
    then obtain m1 m2 where p:m1∈hyper_set m2∈hyper_set m=⟨m1,m2⟩ by
auto
    then obtain x1 x2 where m:m1=hyper_rel{x1} m2=hyper_rel{x2} m=⟨m1,m2⟩
      x1∈Pi(I,X) x2∈Pi(I,X) unfolding hyper_set_def quotient_def by auto
    {
      fix n assume n:n∈I
      with m have ⟨x1n,x2n⟩∈X(n)×X(n) using apply_type by auto
      then have ⟨x1n,x2n⟩∈{⟨i,X(i)×X(i)⟩. i∈I}n using apply_equality[OF
_ f] n by auto
    }
    then have I={n∈I. ⟨x1n,x2n⟩∈{⟨i,X(i)×X(i)⟩. i∈I}n} by auto
    then have {n∈I. ⟨x1n,x2n⟩∈{⟨i,X(i)×X(i)⟩. i∈I}n}:𝔉 using
        ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    then have ⟨x1,x2⟩∈{⟨p,q⟩∈Pi(I,X)×Pi(I,X). {n∈I. ⟨pn,qn⟩∈{⟨i,X(i)×X(i)⟩.
i∈I}n}∈𝔉}
```

```
      using m(4,5) by auto
    with m(1,2) have ⟨m1,m2⟩∈{⟨hyper_rel{x1},hyper_rel{x2}⟩. ⟨x1,x2⟩∈{⟨p,q⟩∈Pi(I,X)×Pi(I,X).
{n∈I. ⟨pn,qn⟩∈{⟨i,X(i)×X(i)⟩. i∈I}n}∈𝔉}}
      by auto
    with p(3) have m∈internal_rel({⟨i,X(i)×X(i)⟩. i∈I})
      unfolding internal_rel_def[OF f] by auto
  }
  then show hyper_set×hyper_set ⊆ internal_rel({⟨i,X(i)×X(i)⟩. i∈I})
by auto
qed
```

91.2 Internal functions

```
definition internal_fun where
S1:Pi(I,λi. Pow(X(i))) ⟹ S2:Pi(I,λi. Pow(X(i))) ⟹ S:Pi(I, λi. S1i
→ S2i)
  ⟹ internal_fun(S) ≡ internal_rel(S)
```

```
lemma internal_fun_is_fun:
  assumes S1:Pi(I,λi. Pow(X(i))) S2:Pi(I,λi. Pow(X(i))) S:Pi(I, λi. S1i
→ S2i)
  shows internal_fun(S):internal_set(S1)→internal_set(S2) S:Pi(I, λi.
Pow(X(i)×X(i))) unfolding Pi_def function_def
proof(safe)
  have SS:S:Pi(I, λi. Pow(X(i)×X(i)))
  proof-
    {
      fix x assume x∈S
      with assms(3) have x∈(∑i∈I. S1i → S2i) unfolding Pi_def by auto
      then obtain i f where f:x=⟨i,f⟩ i∈I f∈S1i → S2i by auto
      from f(3) have f ⊆ S1i×S2i unfolding Pi_def by auto
      then have f ⊆ X(i)×X(i) using apply_type[OF assms(1) f(2)] apply_type[OF
assms(2) f(2)] by auto
      with f(1,2) have x∈(∑i∈I. Pow(X(i)×X(i))) by auto
    }
    then have S ⊆ (∑i∈I. Pow(X(i)×X(i))) by auto
    then show thesis using assms(3) unfolding Pi_def by auto
  qed
  from SS show ⋀x. x ∈ S ⟹ x ∈ (∑i∈I. Pow(X(i) × X(i))) unfold-
ing Pi_def by auto
  from SS show ⋀x. x ∈ I ⟹ x ∈ domain(S) unfolding Pi_def by auto
  from SS show ⋀x y y'. ⟨x, y⟩ ∈ S ⟹ ⟨x, y'⟩ ∈ S ⟹ y = y' unfold-
ing Pi_def function_def by blast
  fix x assume x:x∈internal_fun(S)
  then obtain y z where f:x= ⟨hyper_rel{y},hyper_rel{z}⟩ y:Pi(I,X) z:Pi(I,X)
{n∈I. ⟨yn, zn⟩:Sn}∈𝔉
    unfolding internal_fun_def[OF assms] internal_rel_def[OF SS] by auto
  {
```

```
    fix n assume n:n:{n∈I. ⟨yn, zn⟩:Sn}
    then have n:n:I ⟨yn,zn⟩:Sn by auto
    moreover from n(1) have Sn∈ S1n → S2n using apply_type[OF assms(3)]
by auto
    with n(2) have yn:S1n zn:S2n unfolding Pi_def by auto
    with n(1) have n∈{n:I. yn:S1n} n∈{n:I. zn:S2n} by auto
  }
  then have s:{n∈I. ⟨yn, zn⟩:Sn} ⊆ {n:I. yn:S1n} {n∈I. ⟨yn, zn⟩:Sn} ⊆
{n:I. zn:S2n} by auto
  from f(4) have R:⋀A. A∈Pow(I) ⟹ {n∈I. ⟨yn, zn⟩:Sn} ⊆ A ⟹ A∈𝔉 us-
ing ultraFilter unfolding IsFilter_def IsUltrafilter_def
    by auto
  from R[of {n:I. yn:S1n}] s(1) have {n:I. yn:S1n}∈𝔉 by auto
  then have hyper_rel{y}:internal_set(S1) unfolding internal_set_def[OF
assms(1)]
    using f(2) by auto moreover
  from R[of {n:I. zn:S2n}] s(2) have {n:I. zn:S2n}∈𝔉 by auto
  then have hyper_rel{z}:internal_set(S2) unfolding internal_set_def[OF
assms(2)]
    using f(3) by auto moreover note f(1)
  ultimately show x∈internal_set(S1)×internal_set(S2) by auto
next
  have SS:S:Pi(I, λi. Pow(X(i)×X(i)))
  proof-
    {
      fix x assume x∈S
      with assms(3) have x∈(∑i∈I. S1i → S2i) unfolding Pi_def by auto
      then obtain i f where f:x=⟨i,f⟩ i∈I f∈S1i → S2i by auto
      from f(3) have f ⊆ S1i×S2i unfolding Pi_def by auto
      then have f ⊆ X(i)×X(i) using apply_type[OF assms(1) f(2)] apply_type[OF
assms(2) f(2)] by auto
      with f(1,2) have x∈(∑i∈I. Pow(X(i)×X(i))) by auto
    }
    then have S ⊆ (∑i∈I. Pow(X(i)×X(i))) by auto
    then show thesis using assms(3) unfolding Pi_def by auto
  qed
  fix x assume x∈internal_set(S1)
  then obtain y where x:x=hyper_rel{y} y:Pi(I,X) {n:I. yn∈S1n}:𝔉
    unfolding internal_set_def[OF assms(1)] by auto
  let z={⟨i,if yi∈S1i then (Si)(yi) else yi⟩. i∈I}
  have z:z∈Pi(I,X) unfolding Pi_def function_def apply auto prefer 2
    using x(2) apply_type apply simp
  proof-
    fix i assume i:i:I y  i ∈ S1  i
    from i(1) assms(3) have Si:S1i → S2i using apply_type by auto
    with i(2) have (Si)(yi):S2i using apply_type by auto
    with i(1) show (Si)(yi):X(i) using apply_type[OF assms(2)] by auto
  qed
  {
```

```
    fix n assume n∈{n:I. yn∈S1n}
    then have n:n:I yn:S1n by auto
    from assms(3) n(1) have Sn∈S1n → S2n using apply_type by auto
    with n(2) have ⟨yn,(Sn)(yn)⟩∈Sn using apply_Pair by auto
    with n(1) have n∈{n:I.  ⟨yn, zn⟩:Sn} using n(2) apply_equality[OF
_ z] by auto
  }
  then have {n:I. yn∈S1n} ⊆ {n:I.  ⟨yn, zn⟩:Sn} by auto
  moreover have ∀C∈Pow(I). {n:I. yn∈S1n} ⊆ C ⟶ C ∈ 𝔉
    using ultraFilter x(3) unfolding IsFilter_def IsUltrafilter_def by
auto
  then have {n:I.  ⟨yn, zn⟩:Sn}:Pow(I) ⟶ ({n:I. yn∈S1n} ⊆ {n:I.  ⟨yn,
zn⟩:Sn} ⟶ {n:I.  ⟨yn, zn⟩:Sn} ∈ 𝔉)
    unfolding Ball_def by auto
  then have {n:I. yn∈S1n} ⊆ {n:I.  ⟨yn, zn⟩:Sn} ⟶ {n:I.  ⟨yn, zn⟩:Sn}
∈ 𝔉 by auto
  ultimately have {n:I.  ⟨yn, zn⟩:Sn} ∈ 𝔉 by auto
  then have ⟨hyper_rel{y},hyper_rel{z}⟩∈internal_fun(S)
    using z x(2,3) unfolding internal_fun_def[OF assms] internal_rel_def[OF
SS] by auto
  then show x∈domain(internal_fun(S)) using x(1) unfolding domain_def
by auto
next
  have SS:S:Pi(I, λi. Pow(X(i)×X(i)))
  proof-
    {
      fix x assume x∈S
      with assms(3) have x∈(∑i∈I. S1i → S2i) unfolding Pi_def by auto
      then obtain i f where f:x=⟨i,f⟩ i∈I f∈S1i → S2i by auto
      from f(3) have f ⊆ S1i×S2i unfolding Pi_def by auto
      then have f ⊆ X(i)×X(i) using apply_type[OF assms(1) f(2)] apply_type[OF
assms(2) f(2)] by auto
      with f(1,2) have x∈(∑i∈I. Pow(X(i)×X(i))) by auto
    }
    then have S ⊆ (∑i∈I. Pow(X(i)×X(i))) by auto
    then show thesis using assms(3) unfolding Pi_def by auto
  qed
  fix x y z assume as:⟨x,y⟩∈internal_fun(S) ⟨x,z⟩∈internal_fun(S)
  from as(1) obtain h j where f:x= hyper_rel{h} y=hyper_rel{j} h:Pi(I,X)
j:Pi(I,X) {n∈I. ⟨hn, jn⟩:Sn}∈𝔉
    unfolding internal_fun_def[OF assms] internal_rel_def[OF SS] by auto
  from as(2) obtain k m where g:x=hyper_rel{m} z=hyper_rel{k} m:Pi(I,X)
k:Pi(I,X) {n∈I. ⟨mn, kn⟩:Sn}∈𝔉
    unfolding internal_fun_def[OF assms] internal_rel_def[OF SS] using
f(1) by auto
  from f(1) g(1) have hyper_rel{h} = hyper_rel{m} by auto
  then have ⟨h,m⟩∈hyper_rel using same_image_equiv[OF hyper_equiv] g(3)
by auto
  then have {n:I. hn = mn}:𝔉 unfolding hyper_rel_def by auto
```

```
  with f(5) have {n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}:𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
  with g(5) have {n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}∩{n∈I. ⟨mn, kn⟩:Sn}:𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
  then have ∀A∈Pow(I). {n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}∩{n∈I. ⟨mn, kn⟩:Sn}
⊆ A ⟶ A∈𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
  then have {n:I. jn=kn}:Pow(I) ⟹ {n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}∩{n∈I.
⟨mn, kn⟩:Sn} ⊆ {n:I. jn=kn} ⟶ {n:I. jn=kn}∈𝔉
    by auto
  then have {n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}∩{n∈I. ⟨mn, kn⟩:Sn} ⊆ {n:I.
jn=kn} ⟶ {n:I. jn=kn}∈𝔉 by auto moreover
  {
    fix n assume n:{n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}∩{n∈I. ⟨mn, kn⟩:Sn}
    then have n:n∈I hn=mn ⟨hn, jn⟩:Sn ⟨mn, kn⟩:Sn by auto
    with apply_type[OF assms(3) n(1)] have jn=kn unfolding Pi_def function_def
by force
    with n(1) have n:{n:I. jn=kn} by auto
  }
  then have {n:I. hn = mn}∩{n∈I. ⟨hn, jn⟩:Sn}∩{n∈I. ⟨mn, kn⟩:Sn} ⊆ {n:I.
jn=kn} by auto
  ultimately have {n:I. jn=kn}∈𝔉 by auto
  with f(4) g(4) have ⟨j,k⟩∈hyper_rel unfolding hyper_rel_def by auto
  with f(2) g(2) show y=z using equiv_class_eq[OF hyper_equiv] by auto
qed

lemma internal_fun_apply:
  assumes S1 ∈ (∏i∈I. Pow(X(i))) S2 ∈ (∏i∈I. Pow(X(i))) S ∈ (∏i∈I.
S1  i → S2  i)
    and x∈Pi(I,X) {i:I. xi∈S1i}∈𝔉
  shows internal_fun(S)(hyper_rel{x}) = hyper_rel{{⟨i, if x  i ∈ S1
i then S  i  (x  i) else x  i⟩ . i ∈ I}}
    and {⟨i, if x  i ∈ S1  i then S  i  (x  i) else x  i⟩ . i ∈ I}:Pi(I,X)
proof-
  have SS:S:Pi(I, λi. Pow(X(i)×X(i)))
  proof-
    {
      fix x assume x∈S
      with assms(3) have x∈(∑i∈I. S1i → S2i) unfolding Pi_def by auto
      then obtain i f where f:x=⟨i,f⟩ i∈I f∈S1i → S2i by auto
      from f(3) have f ⊆ S1i×S2i unfolding Pi_def by auto
      then have f ⊆ X(i)×X(i) using apply_type[OF assms(1) f(2)] apply_type[OF
assms(2) f(2)] by auto
      with f(1,2) have x∈(∑i∈I. Pow(X(i)×X(i))) by auto
    }
    then have S ⊆ (∑i∈I. Pow(X(i)×X(i))) by auto
    then show thesis using assms(3) unfolding Pi_def by auto
  qed
  let z={⟨i, if xi∈S1i then (Si)(xi) else xi⟩. i∈I}
```

```
  show z:z∈Pi(I,X) unfolding Pi_def function_def apply auto prefer 2
    using assms(4) apply_type apply simp
  proof-
    fix i assume i:i:I x  i ∈ S1  i
    from i(1) assms(3) have Si:S1i → S2i using apply_type by auto
    with i(2) have (Si)(xi):S2i using apply_type by auto
    with i(1) show (Si)(xi):X(i) using apply_type[OF assms(2)] by auto
  qed
  have hyper_rel{x}∈internal_set(S1) unfolding internal_set_def[OF assms(1)]
    using assms(4,5) by auto
  then have ⟨hyper_rel{x},internal_fun(S)(hyper_rel{x})⟩∈internal_fun(S)
    using apply_Pair internal_fun_is_fun[OF assms(1-3)] by auto
  then have ⟨hyper_rel{x},internal_fun(S)(hyper_rel{x})⟩∈{⟨hyper_rel{x},hyper_rel{y}⟩.
    ⟨x,y⟩ ∈ {⟨p,q⟩ ∈ (∏i∈I. X(i)) × (∏i∈I. X(i)) . {n ∈ I . ⟨p  n,
q  n⟩ ∈ S  n} ∈ 𝔉}}
    unfolding internal_fun_def[OF assms(1-3)] internal_rel_def[OF SS]
by auto
  then obtain t y where A:hyper_rel{x}=hyper_rel{t} internal_fun(S)(hyper_rel{x})
= hyper_rel{y}
    t:Pi(I,X) y:Pi(I,X) {n ∈ I . ⟨t  n, y  n⟩ ∈ S  n} ∈ 𝔉 by auto
  from A(1,3) assms(4) have ⟨x,t⟩∈hyper_rel using eq_equiv_class hyper_equiv
by auto
  then have {i:I. xi = ti}:𝔉 unfolding hyper_rel_def by auto
  with A(5) have {i:I. xi = ti}∩{n ∈ I . ⟨t  n, y  n⟩ ∈ S  n}:𝔉 using
ultraFilter
    unfolding IsFilter_def IsUltrafilter_def by auto moreover
  have {n ∈ I . ⟨x  n, y  n⟩ ∈ S  n}∈Pow(I) by auto
  ultimately have {i:I. xi = ti}∩{n ∈ I . ⟨t  n, y  n⟩ ∈ S  n} ⊆ {n ∈
I . ⟨x  n, y  n⟩ ∈ S  n} ⟶ {n ∈ I . ⟨x  n, y  n⟩ ∈ S  n}:𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
  {
    fix i assume i∈{i:I. xi = ti}∩{n ∈ I . ⟨t  n, y  n⟩ ∈ S  n}
    then have i: I xi=ti ⟨t  i, y  i⟩ ∈ S  i by auto
    then have i:I ⟨xi,yi⟩∈Si by auto
    then have i∈{n ∈ I . ⟨x  n, y  n⟩ ∈ S  n} by auto
  }
  then have {i:I. xi = ti}∩{n ∈ I . ⟨t  n, y  n⟩ ∈ S  n} ⊆ {n ∈ I .
⟨x  n, y  n⟩ ∈ S  n} by auto
  ultimately have {n ∈ I . ⟨x  n, y  n⟩ ∈ S  n}:𝔉 by auto
  moreover have {n∈I. zn = yn}∈Pow(I) by auto
  ultimately have {n ∈ I . ⟨x  n, y  n⟩ ∈ S  n} ⊆ {n∈I. zn = yn} ⟶
{n∈I. zn = yn}∈𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def
    by auto moreover
  {
    fix n assume n:{n ∈ I . ⟨x  n, y  n⟩ ∈ S  n}
    then have n:n∈I ⟨xn,yn⟩∈Sn by auto
    from n(1) have Sn:S1n→S2n using apply_type[OF assms(3)] by auto
```

with n(2) have ⟨xn,yn⟩∈Sn xn∈S1n (Sn)(xn) = yn
using apply_equality[of xn yn Sn S1n λ_. S2n] unfolding Pi_def by auto
then have (if xn∈S1n then (Sn)(xn) else xn) = yn by auto
with n(1) have n∈{n∈I. zn = yn} using apply_equality z by auto
}
then have {n ∈ I . ⟨x n, y n⟩ ∈ S n} ⊆ {n∈I. zn = yn} by auto
ultimately have {n∈I. zn = yn}∈𝔉 by auto
then have ⟨z,y⟩∈hyper_rel unfolding hyper_rel_def using A(4) z by auto
then have ⟨y,z⟩∈hyper_rel using hyper_sym unfolding sym_def by auto
then have hyper_rel{y} = hyper_rel{z} using equiv_class_eq[OF hyper_equiv]
by auto
with A(2) show internal_fun(S)(hyper_rel{x}) = hyper_rel{{⟨i, if x i ∈ S1 i then S i (x i) else x i⟩ . i ∈ I}} by auto
qed

lemma internal_fun_apply_2:
assumes S1 ∈ (∏i∈I. Pow(X(i))) S2 ∈ (∏i∈I. Pow(X(i))) S ∈ (∏i∈I. S1 i → S2 i)
and hyper_rel{x}∈internal_set(S1)
shows internal_fun(S)(hyper_rel{x}) = hyper_rel{{⟨i, if x i ∈ S1 i then S i (x i) else x i⟩ . i ∈ I}}
and {⟨i, if x i ∈ S1 i then S i (x i) else x i⟩ . i ∈ I}:Pi(I,X)
apply (rule internal_fun_apply) using assms(1) apply simp using assms(2) apply simp
using assms(3) apply simp prefer 3 apply (rule internal_fun_apply)
using assms(1) apply simp using assms(2) apply simp
using assms(3) apply simp
proof-
from assms(4) obtain y where A:hyper_rel{x} = hyper_rel{y} y:Pi(I,X) {n∈I. yn∈S1n}∈𝔉
unfolding internal_set_def[OF assms(1)] by auto
from eq_equiv_class[OF A(1) hyper_equiv A(2)] show x:Pi(I,X) unfolding hyper_rel_def by auto
then show x∈Pi(I,X) by auto
from eq_equiv_class[OF A(1) hyper_equiv A(2)] have {n∈I. xn = yn}:𝔉 unfolding hyper_rel_def by auto
with A(3) have {n∈I. xn = yn} ∩{n∈I. yn∈S1n}∈𝔉 using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover have {n∈I. xn∈S1n}∈Pow(I) by auto
ultimately have {n∈I. xn = yn} ∩{n∈I. yn∈S1n} ⊆ {n∈I. xn∈S1n} ⟶ {n∈I. xn∈S1n}:𝔉
using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
then show {n∈I. xn∈S1n}:𝔉 by auto
then show {n∈I. xn∈S1n}:𝔉 by auto
qed

lemma internal_fun_inj:
assumes S1 ∈ (∏i∈I. Pow(X(i))) S2 ∈ (∏i∈I. Pow(X(i))) S ∈ (∏i∈I.

```
S1  i → S2  i)
    and {i:I. Si:inj(S1i,S2i)}∈𝔉
  shows internal_fun(S)∈inj(internal_set(S1), internal_set(S2))
  unfolding inj_def
proof(safe)
  from internal_fun_is_fun[OF assms(1-3)] show internal_fun(S) ∈ internal_set(S1)
→ internal_set(S2) by auto
  fix w x assume as:x∈internal_set(S1) w∈internal_set(S1) internal_fun(S)w
= internal_fun(S)x
  from as(1,2) obtain xx wx where p:xx:Pi(I,X) wx∈Pi(I,X) {i∈I. xxi
∈S1i}∈𝔉  {i∈I. wxi ∈S1i}∈𝔉
    x=hyper_rel{xx} w=hyper_rel{wx}
    unfolding internal_set_def[OF assms(1)] by auto
  from as have ap:hyper_rel
    {{⟨i, if xx  i ∈ S1  i then S  i  (xx  i) else xx  i⟩ . i ∈ I}} =
hyper_rel  {{⟨i, if wx  i ∈ S1  i then S  i  (wx  i) else wx  i⟩ . i ∈
I}} using internal_fun_apply_2[OF assms(1-3)] p(5,6)
    by auto moreover
  have ff:{⟨i, if wx  i ∈ S1  i then S  i  (wx  i) else wx  i⟩ . i ∈ I}∈Pi(I,X)
    {⟨i, if xx  i ∈ S1  i then S  i  (xx  i) else xx  i⟩ . i ∈ I}∈Pi(I,X)
    using apply_type[OF assms(1)] apply_type[OF apply_type[OF assms(3)]]
unfolding Pi_def function_def
    apply auto prefer 2 using apply_type[OF p(2)] apply simp
    using apply_type[OF assms(2)] apply blast
     prefer 2 using apply_type[OF p(1)] apply simp
    using apply_type[OF assms(2)] by blast
  ultimately have ⟨{⟨i, if xx  i ∈ S1  i then S  i  (xx  i) else xx  i⟩
. i ∈ I}, {⟨i, if wx  i ∈ S1  i then S  i  (wx  i) else wx  i⟩ . i ∈ I}⟩∈hyper_rel
    using eq_equiv_class[OF _ hyper_equiv, of {⟨i, if xx  i ∈ S1  i then
S  i  (xx  i) else xx  i⟩ . i ∈ I} {⟨i, if wx  i ∈ S1  i then S  i  (wx
i) else wx  i⟩ . i ∈ I}]
    by auto
  then have Q:{i∈I. {⟨i, if xx  i ∈ S1  i then S  i  (xx  i) else xx
i⟩ . i ∈ I}i = {⟨i, if wx  i ∈ S1  i then S  i  (wx  i) else wx  i⟩ .
i ∈ I}i}∈𝔉
    unfolding hyper_rel_def by auto
  have R:⋀x y. x∈𝔉 ⟹ y∈𝔉 ⟹ x∩y∈𝔉 using ultraFilter unfolding IsFilter_def
IsUltrafilter_def by auto
  from p(3,4) have {i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∈𝔉 using ultraFilter
    unfolding IsFilter_def IsUltrafilter_def by auto
  then have ⋀Q. Q:𝔉 ⟹ {i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∩Q∈𝔉 using ultraFilter
    unfolding IsFilter_def IsUltrafilter_def by auto
  with Q have {i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∩{i∈I. {⟨i, if xx  i ∈
S1  i then S  i  (xx  i) else xx  i⟩ . i ∈ I}i = {⟨i, if wx  i ∈ S1  i
then S  i  (wx  i) else wx  i⟩ . i ∈ I}i}∈𝔉
    by auto
  with assms(4) R have ({i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∩{i∈I. {⟨i, if
xx  i ∈ S1  i then S  i  (xx  i) else xx  i⟩ . i ∈ I}i = {⟨i, if wx  i
∈ S1  i then S  i  (wx  i) else wx  i⟩ . i ∈ I}i})∩{i ∈ I . S  i ∈ inj(S1
```

```
i, S2  i)}:𝔉
    by auto
  then have ⋀Q. Q∈Pow(I) ⟹ {i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∩{i∈I.
{⟨i, if xx  i ∈ S1  i then S  i  (xx  i) else xx  i⟩ . i ∈ I}i = {⟨i,
if wx  i ∈ S1  i then S  i  (wx  i) else wx  i⟩ . i ∈ I}i}∩{i ∈ I . S
i ∈ inj(S1  i, S2  i)} ⊆ Q ⟹ Q:𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
  {
    fix i assume i∈{i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∩{i∈I. {⟨i, if xx
i ∈ S1  i then S  i  (xx  i) else xx  i⟩ . i ∈ I}i = {⟨i, if wx  i ∈ S1
i then S  i  (wx  i) else wx  i⟩ . i ∈ I}i}∩{i ∈ I . S  i ∈ inj(S1  i,
S2  i)}
    then have i:i∈I xxi∈S1i wxi∈S1i S  i  (xx  i) = S  i  (wx  i) Si∈inj(S1i,S2i)
using apply_equality ff by auto
    from i(2-5) have xxi = wxi unfolding inj_def by auto
    with i(1) have i∈{i∈I. xxi = wxi} by auto
  }
  then have {i∈I. xxi ∈S1i}∩{i∈I. wxi ∈S1i}∩{i∈I. {⟨i, if xx  i ∈ S1
i then S  i  (xx  i) else xx  i⟩ . i ∈ I}i = {⟨i, if wx  i ∈ S1  i then
S  i  (wx  i) else wx  i⟩ . i ∈ I}i}∩{i ∈ I . S  i ∈ inj(S1  i, S2  i)}
⊆ {i∈I. xxi = wxi} by auto
  moreover have {i∈I. xxi = wxi}∈Pow(I) by auto
  ultimately have {i∈I. xxi = wxi}:𝔉 using ultraFilter unfolding IsFilter_def
IsUltrafilter_def
    by auto
  then have ⟨xx,wx⟩∈hyper_rel unfolding hyper_rel_def using p(1,2) by
auto
  then show w = x using equiv_class_eq[OF hyper_equiv, THEN sym] p(5,6)
by auto
qed

lemma internal_inj_fun:
  assumes S1 ∈ (∏i∈I. Pow(X(i))) S2 ∈ (∏i∈I. Pow(X(i))) S ∈ (∏i∈I.
S1  i → S2  i)
    and internal_fun(S)∈inj(internal_set(S1), internal_set(S2))
    and Pi({n∈I. Sn∉inj(S1n,S2n)},λn. {⟨q1,q2⟩∈S1n×S1n. (Sn)q1 = (Sn)q2
∧ q1≠q2})≠0
    and Pi(I,X)≠0
  shows {n∈I. Sn∈inj(S1n,S2n)}∈𝔉
proof-
  {
    assume I-{n∈I. Sn∈inj(S1n,S2n)}∈𝔉
    moreover have I-{n∈I. Sn∈inj(S1n,S2n)} = {n∈I. Sn∉inj(S1n,S2n)}
by auto
    ultimately have B:{n∈I. Sn∉inj(S1n,S2n)}∈𝔉 by auto
    from assms(5) obtain q where q:q∈Pi({n∈I. Sn∉inj(S1n,S2n)},λn. {⟨q1,q2⟩∈S1n×S1n.
(Sn)q1 = (Sn)q2 ∧ q1≠q2})
      by auto
```

```
    from assms(6) obtain x where x:x∈Pi(I,X) by auto
    let x={⟨n,if Sn∉inj(S1n,S2n) then fst(qn) else xn⟩. n∈I}
    let y={⟨n,if Sn∉inj(S1n,S2n) then snd(qn) else xn⟩. n∈I}
    have fun:x∈Pi(I,X) y∈Pi(I,X) unfolding Pi_def function_def apply
auto
      prefer 2 using apply_type[OF x] apply simp
        prefer 3 using apply_type[OF x] apply simp
    proof-
      fix n assume ni:n∈I S  n ∉ inj(S1  n, S2  n)
      then have qn∈{⟨q1,q2⟩∈S1n×S1n. (Sn)q1 = (Sn)q2 ∧ q1≠q2}
        using apply_type[OF q] by auto
      then have A:fst(qn)∈S1n snd(qn)∈S1n by auto
      from A(1) show fst(qn)∈X(n) using apply_type[OF assms(1)] ni(1)
by auto
      from A(2) show snd(qn)∈X(n) using apply_type[OF assms(1)] ni(1)
by auto
    qed
    {
      fix n assume n1:n∈{n∈I. Sn∉inj(S1n,S2n)}
      then have n:n∈I Sn∉inj(S1n,S2n) by auto
      have qn∈{⟨q1,q2⟩∈S1n×S1n. (Sn)q1 = (Sn)q2 ∧ q1≠q2}
        using apply_type[OF q n1] by auto
      then obtain qx qy where qq:qn = ⟨qx,qy⟩ qx:S1n qy:S1n (Sn)qx = (Sn)qy
qx≠qy by auto
      from fun have xn=qx yn=qy using n(2) qq(1) apply_equality[of _
_ x] apply_equality[of _ _ y]
        n(1) by auto
      with qq(1-3) have xn:S1n yn:S1n by auto
      with n(1) have n∈{n∈I. xn:S1n} n∈{n∈I. yn:S1n} by auto
    }
    then have {n∈I. Sn∉inj(S1n,S2n)} ⊆ {n∈I. xn:S1n} {n∈I. Sn∉inj(S1n,S2n)}
⊆ {n∈I. yn:S1n} by auto
    moreover note B moreover have {n∈I. xn:S1n}:Pow(I)  {n∈I. yn:S1n}:Pow(I)
by auto
    ultimately have C:{n∈I. xn:S1n}∈𝔉 {n∈I. yn:S1n}∈𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    then have el:hyper_rel{x}∈internal_set(S1) hyper_rel{y}∈internal_set(S1)
      using fun unfolding internal_set_def[OF assms(1)] by auto
    then have els:internal_fun(S)(hyper_rel{x}) = hyper_rel{{⟨i, if x
i ∈ S1  i then S  i  (x  i) else x  i⟩ . i ∈ I}}
      internal_fun(S)(hyper_rel{y}) = hyper_rel{{⟨i, if y  i ∈ S1  i then
S  i  (y  i) else y  i⟩ . i ∈ I}}
      using internal_fun_apply_2[OF assms(1-3)] by auto
    moreover from C have {n∈I. xn:S1n}∩{n∈I. yn:S1n}∈𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    with B have D:{n∈I. Sn∉inj(S1n,S2n)}∩({n∈I. xn:S1n}∩{n∈I. yn:S1n})∈𝔉
using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    from el have fun_els:{⟨i, if x  i ∈ S1  i then S  i  (x  i) else
```

```
x  i⟩ . i ∈ I}∈Pi(I,X)
       {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}∈Pi(I,X)

       using internal_fun_apply_2[OF assms(1-3)] by auto
    {
      fix n assume n1:n∈{n∈I. Sn∉inj(S1n,S2n)}∩{n∈I. xn:S1n}∩{n∈I. yn:S1n}
      then have n:n:I xn:S1n yn:S1n Sn∉inj(S1n,S2n) by auto
      then have R:{⟨i, if x  i ∈ S1  i then S  i  (x  i) else x  i⟩ .
i ∈ I}n = S  n  (x  n)
        {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}n =
S  n  (y  n)
        using apply_equality fun_els by auto
      have qn∈{⟨q1,q2⟩∈S1n×S1n. (Sn)q1 = (Sn)q2 ∧ q1≠q2}
        using apply_type[OF q] n(1,4) by auto
      then obtain qx qy where qq:qn = ⟨qx,qy⟩ qx:S1n qy:S1n (Sn)qx = (Sn)qy
qx≠qy by auto
      from fun have xn=qx yn=qy using n(2,3) qq(1) apply_equality[of
_ _ x] apply_equality[of _ _ y]
        n(1,4) by auto
      with R n(2,3) qq(4,5) have {⟨i, if x  i ∈ S1  i then S  i  (x
i) else x  i⟩ . i ∈ I}n =
        {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}n by
auto
      with n(1) have n:{n∈I. {⟨i, if x  i ∈ S1  i then S  i  (x  i) else
x  i⟩ . i ∈ I}n =
        {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}n} by
auto
    }
    then have {n∈I. Sn∉inj(S1n,S2n)}∩({n∈I. xn:S1n}∩{n∈I. yn:S1n}) ⊆
{n∈I. {⟨i, if x  i ∈ S1  i then S  i  (x  i) else x  i⟩ . i ∈ I}n =
        {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}n} by
auto
    moreover have {n∈I. {⟨i, if x  i ∈ S1  i then S  i  (x  i) else x
i⟩ . i ∈ I}n =
        {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}n}∈Pow(I)
by auto
    moreover note D ultimately have {n∈I. {⟨i, if x  i ∈ S1  i then S
i  (x  i) else x  i⟩ . i ∈ I}n =
        {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}n}∈𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    then have ⟨{⟨i, if x  i ∈ S1  i then S  i  (x  i) else x  i⟩ . i ∈
I},{⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}⟩∈hyper_rel
      using fun_els unfolding hyper_rel_def by auto
    then have hyper_rel{{⟨i, if x  i ∈ S1  i then S  i  (x  i) else x
i⟩ . i ∈ I}} =
      hyper_rel{{⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i
∈ I}}
      using equiv_class_eq[OF hyper_equiv] by auto
    with els have internal_fun(S)(hyper_rel{x}) = internal_fun(S)(hyper_rel{y})
```

```
      by auto
    then have hyper_rel{x} = hyper_rel{y} using assms(4)
      unfolding inj_def[of internal_set(S1) internal_set(S2)] using el
      by auto
    then have ⟨x,y⟩∈hyper_rel using eq_equiv_class[OF _ hyper_equiv] fun
by auto
    then have {n∈I. xn = yn}∈𝔉 unfolding hyper_rel_def by auto
    with B have {n∈I. xn = yn}∩{n∈I. Sn∉inj(S1n,S2n)}∈𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto moreover
    {
      fix n assume n∈{n∈I. xn = yn}∩{n∈I. Sn∉inj(S1n,S2n)}
       then have n:n∈I Sn∉inj(S1n,S2n) xn =yn by auto
      have qn∈{⟨q1,q2⟩∈S1n×S1n. (Sn)q1 = (Sn)q2 ∧ q1≠q2}
        using apply_type[OF q] n by auto
      then obtain qx qy where qq:qn = ⟨qx,qy⟩ qx:S1n qy:S1n (Sn)qx = (Sn)qy
qx≠qy by auto
      from fun have xn=qx yn=qy using n(2) qq(1) apply_equality[of _
_ x] apply_equality[of _ _ y]
        n(1) by auto
      with qq(5) n(3) have False by auto
    }
    then have {n∈I. xn = yn}∩{n∈I. Sn∉inj(S1n,S2n)} = 0 by auto
    ultimately have 0:𝔉 by auto
    then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
  }
  then show {n ∈ I . S  n ∈ inj(S1  n, S2  n)} ∈ 𝔉
    using set_ultrafilter[OF _ ultraFilter] by auto
qed


lemma internal_fun_converse:
  assumes S1 ∈ (∏i∈I. Pow(X(i))) S2 ∈ (∏i∈I. Pow(X(i))) S ∈ (∏i∈I.
S1  i → S2  i)
    and internal_fun(S)∈bij(internal_set(S1), B) and (∏n∈{n ∈ I . S
n ∉ inj(S1  n, S2  n)}.
      {⟨q1,q2⟩ ∈ S1  n × S1  n . S  n  q1 = S  n  q2 ∧ q1 ≠ q2}) ≠ 0
  defines SS ≡ {⟨i, if Si∈inj(S1i,S2i) then converse(Si) else 0⟩. i∈I}
  shows converse(internal_fun(S)) = internal_fun(SS)
  apply(rule fun_extension[of converse(internal_fun(S)) B λ_. internal_set(S1)

      internal_fun(SS) λ_. internal_set(S1)])
proof-
  let S1'= {⟨i,if Si∈inj(S1i,S2i) then S1i else 0⟩. i∈I}
  let S2'= {⟨i,if Si∈inj(S1i,S2i) then S2i else 0⟩. i∈I}
  let S' = {⟨i, if Si∈inj(S1i,S2i) then Si else 0⟩. i∈I}
  from assms(4) have converse(internal_fun(S))∈bij(B,internal_set(S1))
    using bij_converse_bij by auto
  then show CC:converse(internal_fun(S)):B → internal_set(S1) using
```

```
bij_is_fun by auto
  from internal_fun_is_fun[OF assms(1-3)] have F:internal_fun(S):internal_set(S1)→internal
    by auto
  then have INJ:internal_fun(S)∈inj(internal_set(S1),internal_set(S2))
    using bij_is_inj[OF assms(4)] unfolding inj_def by auto
  have S1:S1'∈(∏i∈I. Pow(X(i))) unfolding Pi_def function_def apply
auto
    using if_iff apply auto using apply_type[OF assms(1)] by auto
  have S2:S2'∈(∏i∈I. Pow(X(i))) unfolding Pi_def function_def apply
auto
    using if_iff apply auto using apply_type[OF assms(2)] by auto
  have S:S'∈(∏i∈I. S1'i → S2'i) unfolding Pi_def
    function_def apply auto
  proof-
    {
      fix i x assume as:i∈I x ∈ (if S  i ∈ inj(S1  i, S2  i) then S
i else 0)
      {
        assume A:S  i ∈ inj(S1  i, S2  i)
        with as(2) have x∈Si by auto moreover
        from A have S1'i =S1i using apply_equality[OF _ S1] as(1) by
auto
        moreover from A have S2'i =S2i using apply_equality[OF _ S2]
as(1) by auto
        moreover from as(1) have Si:S1i→S2i using apply_type[OF assms(3)]
by auto
        ultimately have x∈S1'i×S2'i unfolding Pi_def by auto
      } moreover
      {
        assume A:Si∉inj(S1  i, S2  i)
        with as(2) have False by auto
      }
      ultimately show x∈S1'i×S2'i by auto
    }
    {
      fix i x assume as:i ∈ I x ∈ S1'  i S  i ∈ inj(S1  i, S2  i)
      from as(3) have S1'i =S1i using apply_equality[OF _ S1] as(1) by
auto
      with as(2) show x∈domain(Si) using apply_type[OF assms(3)] as(1)
unfolding Pi_def by auto
    }
    {
      fix i x assume as:i ∈ I x ∈ S1'  i
      {
        assume A:S  i ∉ inj(S1  i, S2  i)
        from A have S1'i = 0 using apply_equality[OF _ S1] as(1) by auto
        with as have False by auto
      }
      then show S  i ∈ inj(S1  i, S2  i) by auto
```

```
    }
    {
      fix i x y y' assume as:i ∈ I
       ⟨x, y⟩ ∈ (if S  i ∈ inj(S1  i, S2  i) then S  i else 0)
       ⟨x, y'⟩ ∈ (if S  i ∈ inj(S1  i, S2  i) then S  i else 0)
      {
        assume A:S  i ∉ inj(S1  i, S2  i)
        with as(2) have False by auto
      }
      then have inj:Si∈inj(S1  i, S2  i) by auto
      then have Si:S1i → S2i unfolding inj_def by auto moreover
      from inj as(2,3) have ⟨x,y⟩∈Si ⟨x,y'⟩∈Si by auto
      ultimately show y=y' unfolding Pi_def function_def by blast
    }
  qed
  let S2={⟨i, range(S'  i)⟩. i∈I}
  {
    fix i assume i:i∈I
    then have S'i:S1'i → S2'i using apply_type[OF S] by auto
    then have range(S'i) ⊆ S2'i using func1_1_L5B by auto
    then have range(S'i) ⊆ X(i) using apply_type[OF S2 i] by auto
  }
  then have SS2:S2∈(∏i∈I. Pow(X(i))) unfolding Pi_def function_def by
auto
  have SS:SS∈(∏i∈I. range(S'i) → S1i) unfolding SS_def Pi_def
    function_def apply auto
  proof-
    {
      fix i x assume as:i ∈ I
       x ∈ (if S  i ∈ inj(S1  i, S2  i) then converse(S  i) else 0)
      {
        assume A:S  i ∈ inj(S1  i, S2  i)
        with as(2) have x:converse(Si) by auto
        then obtain x1 x2 where x:⟨x2,x1⟩∈Si x=⟨x1,x2⟩ unfolding converse_def
by auto
        from as(1) have Si:S1i→S2i using apply_type[OF assms(3)] by
auto
        then have Si:S1i→range(Si) using range_of_fun by auto
        with x(1) have x2∈S1i x1∈range(Si) unfolding Pi_def by auto
        with x(2) have x∈range(Si)×S1i by auto
        moreover have S'i = Si using apply_equality[OF _ S, of i] as(1)
A
          by auto
        ultimately have x ∈ range(S'  i) × S1  i by auto
      }moreover
      {
        assume A:S  i ∉ inj(S1  i, S2  i)
        with as(2) have False by auto
      } ultimately
```

```
      show x∈range(S'i)×S1i by auto
    }
    {
      fix i x y assume as:i ∈ I ⟨x, y⟩ ∈ S'  i S  i ∈ inj(S1  i, S2  i)
      from as(1,3) have S' i = Si using apply_equality[OF _ S, of i]
as(1)
        by auto
      with as(2) have ⟨x,y⟩∈Si by auto
      then show y∈range(Si) using rangeI by auto
    }
    {
      fix i x y assume as:i ∈ I ⟨x, y⟩ ∈ S'  i
      {
        assume A:S  i ∉ inj(S1  i, S2  i)
        with as(1,2) have False using apply_equality[OF _ S] by auto
      }
      then show Si:inj(S1  i, S2  i) by auto
    }
    {
      fix i xa y y' assume as:i ∈ I
       ⟨xa, y⟩ ∈ (if S  i ∈ inj(S1  i, S2  i) then converse(S  i) else
0)
       ⟨xa, y'⟩ ∈ (if S  i ∈ inj(S1  i, S2  i) then converse(S  i) else
0)
      {
        assume A:S  i ∉ inj(S1  i, S2  i)
        with as(1,2) have False using apply_equality[OF _ S] by auto
      }
      then have A:Si:inj(S1  i, S2  i) by auto
      with as(2,3) have ⟨xa,y⟩∈converse(Si) ⟨xa,y'⟩∈converse(Si) by auto
      then have ⟨y,xa⟩:Si ⟨y',xa⟩:Si unfolding converse_def by auto
      moreover then have y∈S1i y'∈S1i (Si)y=xa (Si)y'=xa using apply_type[OF
assms(3) as(1)]
        unfolding Pi_def using apply_equality[OF _ inj_is_fun[OF A]] by
auto moreover note A
      ultimately show y=y' unfolding inj_def by auto
    }
  qed
  then have T:SS ∈ (∏i∈I. {⟨i, range(S'  i)⟩ . i ∈ I}  i → S1  i)
    using apply_equality[OF _ SS2] unfolding Pi_def Sigma_def by auto
  from internal_fun_is_fun(1)[OF SS2 assms(1) this]
  have Q:internal_fun(SS) ∈
  internal_set({⟨i, range(S'  i)⟩ . i ∈ I}) → internal_set(S1) by auto
  {
    fix t assume t∈internal_set({⟨i, range(S'  i)⟩ . i ∈ I})
    then obtain x where x:t=hyper_rel{x} x∈Pi(I,X) {n∈I. xn∈{⟨i, range(S'
i)⟩ . i ∈ I}n}:𝔉
      unfolding internal_set_def[OF SS2] by auto
    from x(3) have P:{n∈I. xn∈range(S'  n)}:𝔉 using apply_equality[OF
```

```
_ SS2]
      by auto
    let y={⟨n,if Sn∈inj(S1n,S2n) ∧ xn∈range(Sn) then (converse(Sn)(xn))
else xn⟩. n∈I}
    have Y:y∈Pi(I,X) unfolding Pi_def function_def apply auto
      prefer 2 using apply_type[OF x(2)] apply simp
      prefer 2 using apply_type[OF x(2)] apply simp
    proof-
      fix n xa assume n:n:I Sn:inj(S1n,S2n) ⟨xa, x  n⟩ ∈ S  n
      from n(2) have converse(Sn):range(Sn) → S1n using inj_converse_fun
by auto
      then have converse(Sn)(xn) = xa using apply_equality n(3) converse_iff[of
xn xa]
        by auto
      moreover from n(2,3) have xa∈S1n unfolding inj_def Pi_def by auto
      then have xa∈X(n) using apply_type[OF assms(1)] n(1) by auto
      ultimately show converse(Sn)(xn)∈X(n) by auto
    qed
    {
      fix n assume n∈{n∈I. xn∈range(S'  n)}
      then have n:n:I xn:range(S'n) by auto
      from n(1) have S'n:S1'n→S2'n using apply_type S by auto
      {
        assume Sn∉inj(S1n,S2n)
        then have S'n = 0 using apply_equality[OF _ S] n(1) by auto
        then have range(S'n) = 0 by auto
        with n(2) have False by auto
      }
      then have Z:Sn∈inj(S1n,S2n) by auto
      then have S'n = Sn using n(1) apply_equality[OF _ S] by auto
      with n(2) have n2:xn:range(Sn) by auto
      have yn = (if (S  n ∈ inj(S1  n, S2  n) ∧ x  n ∈ range(S  n)) then
converse(S  n)  (x  n)
         else (x  n)) using apply_equality[of n, OF _ Y] n(1,2) by auto
moreover
      from n2 Z have S  n ∈ inj(S1  n, S2  n) ∧ x  n ∈ range(S  n) by
blast ultimately
      have yn = converse(S  n)  (x  n) by auto
      moreover from Z have converse(Sn):range(Sn) → S1n using inj_converse_fun
by auto
      with n2 have converse(S  n)  (x  n):S1n using apply_type by auto
      ultimately have yn:S1n by auto
      with n(1) have n∈{n∈I. yn:S1n} by auto
    }
    then have Q:{n ∈ I . x  n ∈ range(S'  n)} ⊆ {n∈I. yn:S1n} by auto
    from P have ⋀Q. Q∈Pow(I) ⟹ {n ∈ I . x  n ∈ range(S'  n)} ⊆ Q
⟹ Q:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto moreover
    have {n∈I. yn:S1n}:Pow(I) by auto moreover note Q
```

```
    ultimately have B:{n∈I. yn:S1n}:𝔉 by auto
    with Y have M:hyper_rel{y}∈internal_set(S1) unfolding internal_set_def[OF
assms(1)]
      by auto
    with internal_fun_apply_2[OF assms(1-3)] have A:internal_fun(S)
((hyper_rel)  {y}) =
    hyper_rel
    {{⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}}
      {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}:Pi(I,X)
by auto
    {
      fix t assume t∈{n∈I. xn∈range(S'  n)}∩{n∈I. yn:S1n}
      then have t:t∈I xt:range(S't) yt:S1t by auto
      from A(2) t(1,3) have {⟨i, if y  i ∈ S1  i then S  i  (y  i) else
y  i⟩ . i ∈ I}t =
          (S  t)  (y  t) using apply_equality by auto
      {
        assume St∉inj(S1t,S2t)
        then have S't = 0 using apply_equality[OF _ S] t(1) by auto
        then have range(S't) = 0 by auto
        with t(2) have False by auto
      }
      then have Z:St:inj(S1t,S2t) by auto
      then have ZZ:S't = St using apply_equality[OF _ S] t(1) by auto
      with Z t(1,2) have yt = converse(S  t)  (x  t) using apply_equality[OF
_ Y] by auto
      then have {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i
∈ I}t = (St)(converse(S  t)  (x  t))
        using apply_equality[OF _ A(2)] t(1,3) by auto
      then have {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y  i⟩ . i
∈ I}t = xt
        using right_inverse[OF Z] t(2) ZZ by auto
      with t(1) have t∈{t∈I. {⟨i, if y  i ∈ S1  i then S  i  (y  i) else
y  i⟩ . i ∈ I}t = xt} by auto
    }
    then have {n∈I. xn∈range(S'  n)}∩{n∈I. yn:S1n} ⊆ {t∈I. {⟨i, if y
i ∈ S1  i then S  i  (y  i) else y  i⟩ . i ∈ I}t = xt} by auto
    moreover have {t∈I. {⟨i, if y  i ∈ S1  i then S  i  (y  i) else y
i⟩ . i ∈ I}t = xt}:Pow(I) by auto
    moreover from B P have {n∈I. xn∈range(S'  n)}∩{n∈I. yn:S1n}:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    ultimately have {t∈I. {⟨i, if y  i ∈ S1  i then S  i  (y  i) else
y  i⟩ . i ∈ I}t = xt}:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    with A(2) x(2) have ⟨{⟨i, if y  i ∈ S1  i then S  i  (y  i) else y
i⟩ . i ∈ I},x⟩∈hyper_rel
      unfolding hyper_rel_def by auto
    then have hyper_rel{{⟨i, if y  i ∈ S1  i then S  i  (y  i) else y
i⟩ . i ∈ I}} = hyper_rel{x}
```

```
      using equiv_class_eq[OF hyper_equiv] by auto
    with x(1) have hyper_rel{{⟨i, if y  i ∈ S1  i then S  i  (y  i) else
y  i⟩ . i ∈ I}} = t by auto
    with A(1) have internal_fun(S)(hyper_rel {y}) = t by auto
    then have t∈range(internal_fun(S)) using apply_Pair[OF F M] rangeI
by auto
    then have t∈B using surj_range[OF bij_is_surj[OF assms(4)]] by auto
  }
  then have A:internal_set({⟨i, range(S'  i)⟩ . i ∈ I}) ⊆ B by auto
  {
    fix t assume t∈B
    then have t∈range(internal_fun(S)) using surj_range[OF bij_is_surj[OF
assms(4)]] by auto
    then obtain q where q:⟨q,t⟩∈internal_fun(S) using rangeE by auto
    then have q2:internal_fun(S)q = t using apply_equality[OF _ bij_is_fun[OF
assms(4)]] by auto
    from q have s:q∈internal_set(S1) t∈internal_set(S2) using bij_is_fun[OF
assms(4)] F unfolding Pi_def by auto
    then obtain qx where x:qx∈Pi(I,X) hyper_rel{qx} = q {n∈I. qxn∈S1n}:𝔉
      unfolding internal_set_def[OF assms(1)] by auto
    from s x(2) have f:internal_fun(S)q = hyper_rel{{⟨i,if qx  i ∈ S1
i then S  i  (qx  i) else qx  i⟩. i∈I}}
      {⟨i,if qx  i ∈ S1  i then S  i  (qx  i) else qx  i⟩. i∈I}:Pi(I,X)
      using internal_fun_apply_2[OF assms(1-3), of qx] by auto
    from f(1) q2 have hyper_rel{{⟨i,if qx  i ∈ S1  i then S  i  (qx
i) else qx  i⟩. i∈I}} = t by auto
    moreover from s(2) obtain tx where xx:tx:Pi(I,X) hyper_rel{tx} =
t {n∈I. txn∈S2n}:𝔉
      unfolding internal_set_def[OF assms(2)] by auto
    note xx(2) ultimately have ⟨{⟨i,if qx  i ∈ S1  i then S  i  (qx  i)
else qx  i⟩. i∈I},tx⟩∈hyper_rel
      using eq_equiv_class[OF _ hyper_equiv xx(1)] by auto
    then have {i∈I. {⟨i,if qx  i ∈ S1  i then S  i  (qx  i) else qx
i⟩. i∈I}i=txi}∈𝔉
      unfolding hyper_rel_def by auto
    then have {i∈I. (if qx  i ∈ S1  i then S  i  (qx  i) else qx  i)=txi}∈𝔉
      using f(2) apply_equality by auto
    with x(3) have {n∈I. qxn∈S1n}∩{i∈I. (if qx  i ∈ S1  i then S  i
(qx  i) else qx  i)=txi}∈𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
    from internal_inj_fun[OF assms(1-3) INJ assms(5)] xx(1) have
      {n ∈ I . S  n ∈ inj(S1  n, S2  n)} ∈ 𝔉 by auto ultimately
    have ({n∈I. qxn∈S1n}∩{i∈I. (if qx  i ∈ S1  i then S  i  (qx  i)
else qx  i)=txi})∩{n ∈ I . S  n ∈ inj(S1  n, S2  n)} ∈ 𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
    {
      fix n assume n∈{n∈I. qxn∈S1n}∩{i∈I. (if qx  i ∈ S1  i then S
```

```
i  (qx  i) else qx  i)=txi}∩{n ∈ I . S  n ∈ inj(S1  n, S2  n)}
      then have n1:n∈I qxn:S1n (if qx  n ∈ S1  n then S  n  (qx  n) else
qx  n)=txn Sn∈inj(S1n,S2n) by auto
      from n1(2,3) have S  n  (qx  n) = txn by auto
      then have txn ∈ range(Sn) using rangeI[OF apply_Pair[OF apply_type[OF
assms(3) n1(1)] n1(2)]] by auto
      then have txn∈range(S'n) using apply_equality[of n Sn, OF _ S]
using n1(1,4) by auto
      with n1(1) have n∈{n∈I. txn∈range(S'n)} by auto
      moreover have {⟨i,range(S'i)⟩. i∈I}n = range(S'n) using apply_equality[of
n range(S'n)
            {⟨i,range(S'i)⟩. i∈I} I λ_. {range(S'i). i∈I}] n1(1) unfold-
ing Pi_def function_def
        by auto
      ultimately have n∈{n∈I. txn∈{⟨i,range(S'i)⟩. i∈I}n} by auto
    }
    then have {n∈I. qxn∈S1n}∩{i∈I. (if qx  i ∈ S1  i then S  i  (qx
i) else qx  i)=txi}∩{n ∈ I . S  n ∈ inj(S1  n, S2  n)} ⊆ {n∈I. txn∈{⟨i,range(S'i)⟩.
i∈I}n} by auto
    moreover have {n∈I. txn∈{⟨i,range(S'i)⟩. i∈I}n}:Pow(I) by auto
    ultimately have {n∈I. txn∈{⟨i,range(S'i)⟩. i∈I}n}:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have hyper_rel{tx}∈internal_set({⟨i,range(S'i)⟩. i∈I})
      unfolding internal_set_def[OF SS2] using xx(1) by auto
    with xx(2) have t∈internal_set({⟨i,range(S'i)⟩. i∈I}) by auto
  }
  then have B ⊆ internal_set({⟨i,range(S'i)⟩. i∈I}) by auto
  with A have eq:B = internal_set({⟨i,range(S'i)⟩. i∈I}) by auto
  from subst[OF this[THEN sym], of λB. internal_fun(SS) ∈ B → internal_set(S1)]
Q
  show internal_fun(SS) ∈ B → internal_set(S1) by auto
  {
    fix x assume as:x∈B
    with eq have xx:x∈internal_set({⟨i,range(S'i)⟩. i∈I}) by auto
    then obtain xx where x:xx∈Pi(I,X) x=hyper_rel{xx} {i∈I. xxi∈range(S'i)}:𝔉
      unfolding internal_set_def[OF SS2] using apply_equality[OF _ SS2]
by auto
    have M:internal_fun(SS)x = hyper_rel{{⟨i, if xxi∈{⟨i,range(S'i)⟩.
i∈I}i then (SSi)(xxi) else xxi⟩. i∈I}}
      {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else xxi⟩. i∈I}:Pi(I,X)
      using internal_fun_apply_2[OF SS2 assms(1) T, of xx] xx x(2) by
auto
    have internal_fun(S)(converse(internal_fun(S))x) = x
      using right_inverse_lemma[OF F CC as] by auto
    from as CC have A:converse(internal_fun(S))x :internal_set(S1) us-
ing apply_type by auto
    then obtain yx where yx:yx:Pi(I,X) converse(internal_fun(S))x = hyper_rel{yx}
      {i∈I. yxi :S1i}:𝔉 unfolding internal_set_def[OF assms(1)] by auto
    from CC yx(2) have ⟨x, hyper_rel{yx}⟩∈converse(internal_fun(S))
```

```
      using apply_Pair[OF _ as, of converse(internal_fun(S)) λ_. internal_set(S1)]
by auto
    then have ⟨hyper_rel{yx},x⟩∈internal_fun(S) using converse_iff by
auto
    then have internal_fun(S)(hyper_rel{yx}) = x using apply_equality[OF
_ F] by auto moreover
    have L:internal_fun(S)(hyper_rel{yx}) = hyper_rel{{⟨i, if yxi∈S1i
then (Si)(yxi) else yxi⟩. i∈I}}
      {⟨i, if yxi∈S1i then (Si)(yxi) else yxi⟩. i∈I}∈Pi(I,X)
      using internal_fun_apply_2[OF assms(1-3), of yx] A yx(2) by auto
    ultimately have x = hyper_rel{{⟨i, if yxi∈S1i then (Si)(yxi) else
yxi⟩. i∈I}} by auto
    with x(1,2) have ⟨{⟨i, if yxi∈S1i then (Si)(yxi) else yxi⟩. i∈I},xx⟩∈hyper_rel
      using eq_equiv_class[OF _ hyper_equiv, of _ xx] by auto
    then have {n:I. {⟨i, if yxi∈S1i then (Si)(yxi) else yxi⟩. i∈I}n=xxn}:𝔉
      unfolding hyper_rel_def by auto
    then have {n:I. (if yxn∈S1n then (Sn)(yxn) else yxn)=xxn}:𝔉
      using apply_equality[OF _ L(2)] by auto
    with yx(3) have {n:I. (if yxn∈S1n then (Sn)(yxn) else yxn)=xxn}∩{n∈I.
yxn :S1n}:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
    {
      fix s assume s∈{n:I. (if yxn∈S1n then (Sn)(yxn) else yxn)=xxn}∩{n∈I.
yxn :S1n}
      then have s:s∈I (if yxs∈S1s then (Ss)(yxs) else yxs)=xxs yxs:S1s
by auto
      then have s∈I (Ss)(yxs)=xxs by auto
      then have s∈I (SSs)((Ss)(yxs)) = (SSs)(xxs) by auto
      with s(3) have s∈{s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s}
by auto
    }
    then have {n:I. (if yxn∈S1n then (Sn)(yxn) else yxn)=xxn}∩{n∈I. yxn
:S1n} ⊆ {s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s} by auto
    moreover have {s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s}:Pow(I)
by auto
    ultimately have {s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s}:𝔉
using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have {s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s}∩{i∈I. xxi∈range(S'i)}:𝔉
      using x(3) ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto moreover
    {
      fix t assume t∈{s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s}∩{i∈I.
xxi∈range(S'i)}
      then have t:t:I (SSt)((St)(yxt)) = (SSt)(xxt) xxt∈range(S't) yxt:S1t
by auto
      {
        assume St∉inj(S1t,S2t)
```

```
        then have S't = 0 using t(1) apply_equality[OF _ S] by auto
        then have range(S't) = 0 by auto
        with t(3) have False by auto
      }
      then have INJ:St:inj(S1t,S2t) by auto
      from M(2) have {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi)
else xxi⟩. i∈I}t = (if xxt∈{⟨i,range(S'i)⟩. i∈I}t then (SSt)(xxt) else
xxt)
        using apply_equality t(1) by auto
      then have {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else
xxi⟩. i∈I}t = (if xxt∈range(S't) then (SSt)(xxt) else xxt)
        using apply_equality[OF _ SS2] t(1) by auto
      then have {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else
xxi⟩. i∈I}t = (SSt)(xxt)
        using t(3) by auto
      then have {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else
xxi⟩. i∈I}t = (SSt)((St)(yxt))
        using t(2) by auto
      then have {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else
xxi⟩. i∈I}t = converse(St)((St)(yxt))
        using apply_equality[OF _ SS] t(1) INJ unfolding SS_def by auto
      then have {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else
xxi⟩. i∈I}t = yxt using left_inverse[OF INJ] t(4) by auto
      with t(1) have t∈{t∈I. {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then
(SSi)(xxi) else xxi⟩. i∈I}t = yxt} by auto
    }
    then have {s∈I. (SSs)((Ss)(yxs)) = (SSs)(xxs) ∧ yxs∈S1s}∩{i∈I. xxi∈range(S'i)}
⊆ {t∈I. {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else xxi⟩.
i∈I}t = yxt} by auto
    moreover have {t∈I. {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi)
else xxi⟩. i∈I}t = yxt}:Pow(I) by auto
    ultimately have {t∈I. {⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi)
else xxi⟩. i∈I}t = yxt}:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have ⟨{⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi) else
xxi⟩. i∈I},yx⟩∈hyper_rel
      using M(2) yx(1) unfolding hyper_rel_def by auto
    then have hyper_rel{{⟨i, if xxi∈{⟨i,range(S'i)⟩. i∈I}i then (SSi)(xxi)
else xxi⟩. i∈I}} = hyper_rel{yx}
      using equiv_class_eq[OF hyper_equiv] by auto
    with M(1) have internal_fun(SS)x = hyper_rel{yx} by auto
    with yx(2) have internal_fun(SS)x = converse(internal_fun(S))x by
auto
    then show converse(internal_fun(S))  x = internal_fun(SS)  x by auto
  }
qed

definition isHyperFinite where
H∈Pow(hyper_set) ⟹ isHyperFinite(H) ≡ ∃S∈Pi(I,λi. FinPow(X(i))). H
```

```
= internal_set(S)

lemma hyperfinite_internal:
  assumes H∈Pow(hyper_set) isHyperFinite(H)
  shows isInternal(H)
proof-
  from assms(2) obtain S where S:S:Pi(I,λi. FinPow(X(i))) H=internal_set(S)
    unfolding isHyperFinite_def[OF assms(1)] isInternal_def by auto
  from S(1) have S:Pi(I,λi. Pow(X(i))) unfolding Pi_def FinPow_def by
auto
  with S(2) show thesis unfolding isInternal_def by auto
qed

end
end
```

92 Hypernatural numbers

```
theory HyperNatural_ZF imports UltraConstruction_ZF ZF.Cardinal Finite1
begin
```

The goal of this file is to show that hyperfinite sets are the sets of hypernatural numbers internally bijective with an open left ray of hypernatural.

The cardinality of a non-trivial set difference is strictly smaller for finite sets

```
lemma finite_diff_strict_smaller: assumes Finite(A) A≠0 A∩B≠0
  shows |A-B| < |A|
proof -
  from assms(3) obtain x where x:x∈B x∈A by auto
  from x(1) have x∉A-B by auto moreover note x(2) moreover
  from assms(1) have A∈FinPow(A) unfolding FinPow_def by auto
  then have A-B∈FinPow(A) using subset_finpow[of A A A-B] by auto
  ultimately have |(A-B)∪{x}| = succ(|A-B|) using card_fin_add_one(1)[of
A-B A x] by auto
  then have A:|A-B| < |(A-B)∪{x}| using le_refl[OF Ord_cardinal] by auto
  moreover have (A-B)∪{x} ⊆ A using x(2) by auto
  then have (A-B)∪{x} ≲ A using subset_imp_lepoll by auto
  then have |(A-B)∪{x}| ≤ |A| using well_ord_lepoll_imp_cardinal_le Finite_imp_well_ord[OF
assms(1)]
    by auto
  with A show thesis using lt_trans2 by auto
qed
```

This applies to strict subsets

```
corollary strict_subset_smaller: assumes Finite(A) B ⊆ A B≠A
  shows |B| < |A|
proof-
  {
```

```
    assume a0:A = 0
    with assms(2) have B = 0 by auto
    with a0 assms(3) have False by auto
  }
  then have A≠0 by auto moreover
  have A∩(A-B) ≠0 using assms(2,3) by auto moreover
  note assms(1) ultimately
  have |A-(A-B)| < |A| using finite_diff_strict_smaller by auto
  moreover have A-(A-B) = B using assms(2) by auto
  ultimately show thesis by auto
qed
```

It can be further applied to solve the relation between injective and bijective functions on finite sets.

```
corollary inj_function_same_card_bij: assumes Finite(B) A ≈ B f∈inj(A,B)
  shows f∈bij(A,B)
proof-
  have fA ⊆ B using func1_1_L6(2) inj_is_fun[OF assms(3)] by auto
  with assms(1) have R:fA ≠ B ⟹ |fA| < |B| using strict_subset_smaller
by auto
  from assms(3) have f:bij(A,range(f)) using inj_bij_range by auto
  then have A≈range(f) unfolding eqpoll_def by auto
  then have A≈fA using range_image_domain[OF inj_is_fun[OF assms(3)]]
by auto
  then have |A| = |fA| using cardinal_cong by auto moreover
  have |A| = |B| using cardinal_cong assms(2) by auto ultimately
  have |fA| = |B| by auto
  with R have fA ≠ B ⟹ |B| < |B| by auto
  then have A:fA = B using lt_not_refl by auto
  then show thesis using surj_def_alt assms(3) inj_is_fun unfolding bij_def

    by auto
qed
```

This locale deals with hyper numbers.

```
locale hyperNatural = ultra _ nat λ_. nat +
  assumes non_pricipal_filter:⋂𝔉 = 0

begin

abbreviation hyperNat (*ℕ) where
*ℕ ≡ hyper_set

definition seq_class ([_]) where
x∈nat→nat ⟹ [x] ≡ hyper_rel{x}

definition omega (ω) where
ω = [id(nat)]
```

```
definition incl (*_ 70) where
x∈nat ⟹ *x ≡ [ConstantFunction(nat,x)]

lemma incl_inj_nat:
  shows {⟨x,*x⟩. x∈nat} ∈ inj(nat, *ℕ) using incl_inj[of nat]
    seq_class_def[OF func1_3_L1] incl_def
  unfolding incl_def by auto

lemma omega_not_nat:
  shows x∈nat⟶(*x) ≠ ω and ω:*ℕ
proof
  have id(nat):nat→nat using id_def by auto
  then have [id(nat)]:*ℕ using seq_class_def[of id(nat)]
    unfolding hyper_set_def quotient_def by auto
  then show ω:*ℕ unfolding omega_def by auto
  {
    assume a:x∈nat (*x) = ω
    from a(2) have [ConstantFunction(nat,x)] = [id(nat)]
      unfolding omega_def incl_def[OF a(1)].
    then have ⟨ConstantFunction(nat,x),id(nat)⟩∈hyper_rel
      using same_image_equiv[OF hyper_equiv, of id(nat)] inj_is_fun[OF
id_inj]
      unfolding seq_class_def[OF inj_is_fun[OF id_inj]] seq_class_def[OF
func1_3_L1[OF a(1)]] by auto
    then have {n∈nat. ConstantFunction(nat,x)n = id(nat)n}:𝔉 unfold-
ing hyper_rel_def by auto
    then have {n∈nat. x = n}:𝔉 using apply_equality[of _ _ id(nat) nat
λ_. nat] inj_is_fun[OF id_inj]
      func1_3_L2[of _ nat x] by auto moreover
    have {n∈nat. x = n} = {x} using a(1) by auto ultimately
    have f:{x}:𝔉 by auto
    {
      fix A assume x:x∈A A⊆ nat
      then have {x} ⊆ A by auto
      then have A:𝔉 using f ultraFilter x(2) unfolding IsFilter_def IsUltrafilter_def
by auto
    }
    then have {A∈Pow(nat). x:A} ⊆ 𝔉 by auto moreover
    {
      fix A assume a:A∈𝔉
      with f have y:A∩{x}:𝔉 using ultraFilter unfolding IsFilter_def
IsUltrafilter_def
        by auto
      {
        assume x∉A
        then have A∩{x} = 0 by auto
        with y have 0:𝔉 by auto
        then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
```

```
      }
      then have x:A by auto
      moreover from a have A∈Pow(nat) using ultraFilter unfolding IsFilter_def
IsUltrafilter_def by auto
      ultimately have A∈{A∈Pow(nat). x:A} by auto
    }
    then have 𝔉 ⊆ {A∈Pow(nat). x:A} by auto ultimately
    have 𝔉 = {A∈Pow(nat). x:A} by auto
    then have 𝔉=0 ∨ x∈⋂𝔉 by auto
    with non_pricipal_filter have 𝔉=0 by auto
    then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
  }
  then show x ∈ nat ⟹ (*x) ≠ ω using ultraFilter unfolding IsFilter_def
IsUltrafilter_def by auto
qed
```

We define the order relation as the internal relation created by a constant relation of order in natural numbers

```
definition rel_star (*_* 80) where
R ⊆ nat×nat ⟹ *R* ≡ internal_rel(ConstantFunction(nat,R))

definition lessEq_star (infix *≤ 80) where
x*≤y ≡ ⟨x,y⟩∈(*Le*)

definition less_star (infix *< 80) where
x*<y ≡ ⟨x,y⟩∈(*Lt*)
```

Two hypernaturals are ordered iff where their representative sequences are ordered is a set in the filter

```
lemma star_rel_seq:
  assumes x:nat→nat y:nat→nat R ⊆ nat×nat
  shows ⟨[x],[y]⟩∈*R* ⟷ {i∈nat. ⟨xi,yi⟩∈R}∈𝔉
proof(safe)
have S_fun:ConstantFunction(nat,R):nat → Pow(nat×nat)
    apply (rule func1_3_L1) using assms(3) by auto
  {
    assume ⟨[x],[y]⟩∈*R*
    then have ⟨[x],[y]⟩∈internal_rel(ConstantFunction(nat,R))
      unfolding rel_star_def[OF assms(3)].
    then obtain z t where zt:[x] = [z] [y] = [t] z:nat→nat  t:nat→nat
{n ∈ nat . ⟨z  n, t  n⟩ ∈ ConstantFunction(nat,R)n} ∈ 𝔉
      unfolding internal_rel_def[OF S_fun] using seq_class_def by auto
    from zt(5) have A:{n ∈ nat . ⟨z  n, t  n⟩ ∈ R} ∈ 𝔉
      using func1_3_L2 by auto
    from zt(1) have ⟨x,z⟩∈hyper_rel unfolding seq_class_def[OF zt(3)]
      seq_class_def[OF assms(1)] using eq_equiv_class[OF _ hyper_equiv
zt(3)] by auto
    then have {n:nat. xn = zn}∈𝔉 unfolding hyper_rel_def by auto
```

```
    with A have A:{n ∈ nat . ⟨z  n,t  n⟩∈R}∩{n:nat. xn = zn}∈𝔉 using
ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    from zt(2) have ⟨y,t⟩∈hyper_rel unfolding seq_class_def[OF zt(4)]
      seq_class_def[OF assms(2)] using eq_equiv_class[OF _ hyper_equiv
zt(4)] by auto
    then have {n:nat. yn = tn}∈𝔉 unfolding hyper_rel_def by auto
    with A have {n:nat. yn = tn}∩({n ∈ nat . ⟨z  n,t  n⟩∈R}∩{n:nat. xn
= zn})∈𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have ∀B∈Pow(nat). {n:nat. yn = tn}∩({n ∈ nat . ⟨z  n, t  n⟩∈R}∩{n:nat.
xn = zn}) ⊆ B ⟶ B∈𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
    then have {n∈nat. ⟨xn,yn⟩∈R}∈Pow(nat) ⟹ {n:nat. yn = tn}∩({n ∈
nat . ⟨z  n, t  n⟩∈R}∩{n:nat. xn = zn}) ⊆ {n∈nat. ⟨xn,yn⟩∈R} ⟹ {n∈nat.
⟨xn,yn⟩∈R}∈𝔉
      by auto
    then have {n:nat. yn = tn}∩({n ∈ nat . ⟨z  n, t  n⟩∈R}∩{n:nat. xn
= zn}) ⊆ {n∈nat. ⟨xn,yn⟩∈R} ⟹ {n∈nat. ⟨xn,yn⟩∈R}∈𝔉 by auto moreover
    {
      fix n assume n∈{n:nat. yn = tn}∩({n ∈ nat . ⟨z  n, t  n⟩∈R}∩{n:nat.
xn = zn})
      then have n∈nat yn = tn ⟨z  n, t  n⟩∈R xn = zn by auto
      then have n∈nat ⟨xn,yn⟩∈R by auto
      then have n∈{n∈nat. ⟨xn,yn⟩∈R} by auto
    }
    then have {n:nat. yn = tn}∩({n ∈ nat . ⟨z  n, t  n⟩∈R}∩{n:nat. xn
= zn}) ⊆ {n∈nat. ⟨xn,yn⟩∈R} by auto
    ultimately show {n∈nat. ⟨xn,yn⟩∈R}∈𝔉 by auto
  }
  {
    assume {n∈nat. ⟨xn,yn⟩∈R}∈𝔉
    then have {n ∈ nat . ⟨x  n, y  n⟩ ∈ ConstantFunction(nat,R)n} ∈ 𝔉
      using func1_3_L2 by auto
    with assms have ⟨[x],[y]⟩∈internal_rel(ConstantFunction(nat,R))
      unfolding internal_rel_def[OF S_fun] seq_class_def[OF assms(1)]
seq_class_def[OF assms(2)]
      by auto
    then show ⟨[x],[y]⟩∈*R* unfolding rel_star_def[OF assms(3)].
  }
qed

corollary star_rel_imp_seq:
  assumes x:nat→nat y:nat→nat R1 ⊆ R2 R2 ⊆ nat×nat
  shows ⟨[x],[y]⟩∈*R1* ⟶ ⟨[x],[y]⟩∈*R2*
proof(safe)
  assume ⟨[x],[y]⟩∈*R1*
  then have {n∈nat. ⟨xn,yn⟩∈R1}∈𝔉 using star_rel_seq[of x y R1] assms
by auto
```

```
  moreover have {n∈nat. ⟨xn,yn⟩∈R1} ⊆ {n∈nat. ⟨xn,yn⟩∈R2} using assms(3)
by auto
  moreover note is_filter_def_split(5)[of 𝔉 nat]
  ultimately have {n∈nat. ⟨xn,yn⟩∈R2}∈𝔉 using ultraFilter unfolding IsUltrafilter_def
by blast
  then show ⟨[x],[y]⟩∈*R2* using star_rel_seq[of x y R2] assms(1,2,4)
by auto
qed

corollary star_rel_intersect_seq:
  assumes x:nat→nat y:nat→nat R1 = R2∩R3 R2 ⊆ nat×nat R3 ⊆ nat×nat
  shows ⟨[x],[y]⟩∈*R1* ⟷ ⟨[x],[y]⟩∈*R2* ∧ ⟨[x],[y]⟩∈*R3*
proof(safe)
  assume as:⟨[x],[y]⟩∈*R1*
  with assms(1-4) show ⟨[x],[y]⟩∈*R2* using star_rel_imp_seq[of x y R1
R2] by auto
  from as assms(1-3,5) show ⟨[x],[y]⟩∈*R3* using star_rel_imp_seq[of
x y R1 R3] by auto
next
  assume ⟨[x],[y]⟩∈*R2* ⟨[x],[y]⟩∈*R3*
  then have {n∈nat. ⟨xn,yn⟩∈R2}∈𝔉 {n∈nat. ⟨xn,yn⟩∈R3}∈𝔉
    using star_rel_seq[of x y] assms(1,2,4,5) by auto
  then have {n∈nat. ⟨xn,yn⟩∈R2}∩{n∈nat. ⟨xn,yn⟩∈R3}∈𝔉 using
    is_filter_def_split(4) ultraFilter unfolding IsUltrafilter_def by
auto
  moreover have {n∈nat. ⟨xn,yn⟩∈R2}∩{n∈nat. ⟨xn,yn⟩∈R3} = {n∈nat. ⟨xn,yn⟩∈R2
∧ ⟨xn,yn⟩∈R3} by auto
  then have {n∈nat. ⟨xn,yn⟩∈R2}∩{n∈nat. ⟨xn,yn⟩∈R3} = {n∈nat. ⟨xn,yn⟩∈R2∩R3}
by auto
  with assms(3) have {n∈nat. ⟨xn,yn⟩∈R2}∩{n∈nat. ⟨xn,yn⟩∈R3} = {n∈nat.
⟨xn,yn⟩∈R1} by auto
  ultimately have {n∈nat. ⟨xn,yn⟩∈R1}∈𝔉 by auto
  then show ⟨[x],[y]⟩∈*R1* using star_rel_seq[of x y R1] assms(1-4) by
auto
qed

corollary less_than_seq:
  assumes x:nat→nat y:nat→nat
  shows [x] *≤ [y] ⟷ {i∈nat. xi ≤ yi}∈𝔉
proof -
  have {i ∈ nat. ⟨x  i, y  i⟩ ∈ {⟨p,q⟩ ∈ nat × nat . p ≤ q}} = {i ∈ nat.
xi ≤ yi}
    apply simp using assms apply_type by auto
  then have R1:[x] *≤ [y] ⟷ {i ∈ nat. ⟨x  i, y  i⟩ ∈ {⟨p,q⟩ ∈ nat ×
nat . p ≤ q}}∈𝔉 ⟹ [x] *≤ [y] ⟷ {i∈nat. xi ≤ yi}∈𝔉 by auto
  have [x] *≤ [y] ⟷ {i ∈ nat. ⟨x  i, y  i⟩ ∈ {⟨p,q⟩ ∈ nat × nat . p
≤ q}}∈𝔉  unfolding lessEq_star_def using star_rel_seq[OF assms,
  of {⟨p,q⟩ ∈ nat × nat . p ≤ q}] unfolding Le_def by auto
  with R1 show thesis by auto
```

```
qed

corollary less_seq:
  assumes x:nat→nat y:nat→nat
  shows [x] *< [y] ⟷ {i∈nat. xi <yi}∈𝔉
proof -
  have {i ∈ nat. ⟨x  i, y  i⟩ ∈ {⟨p,q⟩ ∈ nat × nat . p <q}} = {i ∈ nat.
xi < yi}
    apply simp using assms apply_type by auto
  then have R1:[x] *< [y] ⟷ {i ∈ nat. ⟨x  i, y  i⟩ ∈ {⟨p,q⟩ ∈ nat ×
nat . p < q}}∈𝔉 ⟹ [x] *< [y] ⟷ {i∈nat. xi <yi}∈𝔉 by auto
  have [x] *< [y] ⟷ {i ∈ nat. ⟨x  i, y  i⟩ ∈ {⟨p,q⟩ ∈ nat × nat . p
< q}}∈𝔉  unfolding less_star_def using star_rel_seq[OF assms,
  of {⟨p,q⟩ ∈ nat × nat . p < q}] unfolding Lt_def by auto
  with R1 show thesis by auto
qed

corollary less_imp_less_eq:
  assumes x:nat→nat y:nat→nat
  shows [x] *< [y] ⟶ [x] *≤ [y]
    unfolding lessEq_star_def less_star_def
    apply (rule star_rel_imp_seq[OF assms]) using leI unfolding Lt_def
Le_def by auto
```

There is a hypernatural bigger than all natural numbers

```
lemma omega_bigger_naturals:
  assumes x:nat
  shows (*x) *≤ ω
proof-
  from assms have (*x) = [ConstantFunction(nat,x)]
    using incl_def by auto
  have {n∈nat. ConstantFunction(nat,x)n ≤ id(nat)n} = {n∈nat. x ≤ n}
    using func1_3_L2[of _ nat x] id_iff by auto
  have nat-{n∈nat. x ≤ n}∈FinPow(nat) apply (rule nat_induct[of x λq.
nat-{n∈nat. q ≤ n}∈FinPow(nat)])
  proof-
    from assms show x∈nat.
    have {n ∈ nat . 0 ≤ n} = nat using nat_0_le by auto
    then have nat-{n ∈ nat . 0 ≤ n} = 0 by auto
    then show nat-{n ∈ nat . 0 ≤ n}∈FinPow(nat) using Finite_0 unfold-
ing FinPow_def by auto
    {
      fix x assume x:x∈nat nat - {n ∈ nat . x ≤ n} ∈ FinPow(nat)
      {
        fix t assume t:t∈(nat - {n ∈ nat . x ≤ n})∪{x}
        {
          assume e:t=x
          then have t∈nat using x(1) by auto moreover
          {
```

```
          assume succ(x) ≤ x
          then have x < x using succ_leE by auto
          then have False by auto
        } moreover note e
        ultimately have t∈nat-{n ∈ nat. succ(x) ≤ n} by auto
      } moreover
      {
        assume e:t≠x
        with t have t:t∈nat - {n ∈ nat . x ≤ n} by auto
        then have tnat:t∈nat by auto
        {
          assume succ(x)≤t
          then have x<t using succ_leE by auto
          then have x≤t using leI by auto
          with t have False by auto
        }
        with tnat have t∈nat-{n ∈ nat. succ(x) ≤ n} by auto
      } ultimately
      have t∈nat-{n ∈ nat. succ(x) ≤ n} by auto
    }
    then have nat - {n ∈ nat . x ≤ n} ∪ {x} ⊆ nat - {n ∈ nat . succ(x)
≤ n} by auto moreover
    {
      fix t assume t:t∈nat - {n ∈ nat . succ(x) ≤ n}
      {
        assume tn:t≠x
        {
          assume tx:x≤t
          with tn have x < t using le_iff by auto
          then have succ(x)≤t using succ_leI by auto
          with t have False by auto
        }
        with t have t∈nat - {n ∈ nat . x ≤ n} by auto
      }
      then have t∈nat - {n ∈ nat . x ≤ n} ∪ {x} by auto
    }
    then have nat - {n ∈ nat . succ(x) ≤ n} ⊆ nat - {n ∈ nat . x ≤
n} ∪ {x} by auto
    ultimately have nat - {n ∈ nat . succ(x) ≤ n} = nat - {n ∈ nat
. x ≤ n} ∪ {x} by auto
    then show nat - {n ∈ nat . succ(x) ≤ n}∈FinPow(nat)
      using union_finpow[OF x(2) singleton_in_finpow[OF x(1)]] by auto
  }
qed moreover
{
  fix x assume as:x∈nat {x}:𝔉
  {
    fix A assume x:x∈A A⊆ nat
    then have {x} ⊆ A by auto
```

```
      then have A:𝔉 using as(2) ultraFilter x(2) unfolding IsFilter_def
IsUltrafilter_def by auto
    }
    then have {A∈Pow(nat). x:A} ⊆ 𝔉 by auto moreover
    {
      fix A assume a:A∈𝔉
      with as(2) have y:A∩{x}:𝔉 using ultraFilter unfolding IsFilter_def
IsUltrafilter_def
          by auto
      {
        assume x∉A
        then have A∩{x} = 0 by auto
        with y have 0:𝔉 by auto
        then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
      }
      then have x:A by auto
      moreover from a have A∈Pow(nat) using ultraFilter unfolding IsFilter_def
IsUltrafilter_def by auto
      ultimately have A∈{A∈Pow(nat). x:A} by auto
    }
    then have 𝔉 ⊆ {A∈Pow(nat). x:A} by auto ultimately
    have 𝔉 = {A∈Pow(nat). x:A} by auto
    then have 𝔉=0 ∨ x∈⋂𝔉 by auto
    with non_pricipal_filter have 𝔉=0 by auto
    then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
  }
  then have ∀x∈nat. {x} ∉ 𝔉 by auto
  ultimately have nat - {n ∈ nat . x ≤ n}∉𝔉 using ultrafilter_finite_set[OF
ultraFilter]
    by auto
  then have {n ∈ nat . x ≤ n}:𝔉 using ultraFilter set_ultrafilter[of
{n ∈ nat . x ≤ n}] by auto
  then have {n∈nat. ConstantFunction(nat,x)n ≤ id(nat)n}:𝔉
    using func1_3_L2[of _ nat x] by auto
  then have [ConstantFunction(nat,x)]*≤[id(nat)] using less_than_seq[OF
func1_3_L1 inj_is_fun[OF id_inj]]
    assms by auto
  then show thesis unfolding omega_def incl_def[OF assms] by auto
qed
```

Every function on the natural numbers can be extended to a function on the hypernaturals

```
definition hyper_fun (*_*) where
f:nat→nat ⟹ *f* ≡ internal_fun(ConstantFunction(nat,f))

lemma hyper_fun_on_nat:
  assumes f∈nat→nat x∈nat
```

```
  shows *f*(*x) = *(fx)
proof-
  have e:*f*(*x) = internal_fun(ConstantFunction(nat,f)) (hyper_rel{ConstantFunction(nat,x)
    unfolding hyper_fun_def[OF assms(1)] incl_def[OF assms(2)] seq_class_def[OF
func1_3_L1[OF assms(2)]] by auto
  have A:ConstantFunction(nat,nat):nat→Pow(nat) using func1_3_L1 by auto
  have B:ConstantFunction(nat,f):(∏i∈nat. ConstantFunction(nat, nat)
i → ConstantFunction(nat, nat)  i)
    unfolding Pi_def function_def Sigma_def apply auto unfolding ConstantFunction_def
apply auto
      prefer 2 using assms(1) func1_3_L2[of _ nat nat] unfolding Pi_def
ConstantFunction_def apply auto
    unfolding function_def by blast
  have {⟨i,nat⟩. i∈nat} = ConstantFunction(nat,nat) unfolding ConstantFunction_def
by auto
  then have internal_set(ConstantFunction(nat,nat)) = *ℕ using internal_total_set
by auto
  then have C:hyper_rel  {ConstantFunction(nat, x)} ∈
    internal_set(ConstantFunction(nat, nat)) unfolding hyper_set_def quotient_def
    using func1_3_L1[OF assms(2)] by auto
  from e have *f*(*x) =hyper_rel
    {{⟨i, if ConstantFunction(nat, x)  i ∈ ConstantFunction(nat, nat)
i
          then ConstantFunction(nat, f)  i  (ConstantFunction(nat, x)
i)
          else ConstantFunction(nat, x)  i⟩ .
      i ∈ nat}}
    using internal_fun_apply_2[of ConstantFunction(nat,nat) ConstantFunction(nat,nat)
ConstantFunction(nat,f)
        ConstantFunction(nat,x), OF A A B C] by auto
  then have *f*(*x) =hyper_rel
    {{⟨i, if x ∈ nat then f  x else x⟩. i ∈ nat}}
    using func1_3_L2[of _ nat] by auto
  with assms(2) have *f*(*x) =hyper_rel{{⟨i,fx⟩. i∈nat}} by auto
  then show thesis unfolding seq_class_def[OF func1_3_L1[OF apply_type[OF
assms]]] incl_def[OF apply_type[OF assms]]
    unfolding ConstantFunction_def by auto
qed
```

Applying a function extended from the natural numbers to the class of certain sequence gives out the sequence of the composition

```
lemma hyper_fun_on_nat_2:
  assumes f∈nat→nat x∈nat→nat
  shows *f*([x]) = [f O x]
proof-
  have e:*f*([x]) = internal_fun(ConstantFunction(nat,f)) (hyper_rel{x})
    unfolding hyper_fun_def[OF assms(1)] seq_class_def[OF assms(2)] by
auto
  have A:ConstantFunction(nat,nat):nat→Pow(nat) using func1_3_L1 by auto
```

```
  have B:ConstantFunction(nat,f):(∏i∈nat. ConstantFunction(nat, nat)
i → ConstantFunction(nat, nat)  i)
    unfolding Pi_def function_def Sigma_def apply auto unfolding ConstantFunction_def
apply auto
      prefer 2 using assms(1) func1_3_L2[of _ nat nat] unfolding Pi_def
ConstantFunction_def apply auto
    unfolding function_def by blast
  have {⟨i,nat⟩. i∈nat} = ConstantFunction(nat,nat) unfolding ConstantFunction_def
by auto
  then have internal_set(ConstantFunction(nat,nat)) = *ℕ using internal_total_set
by auto
  then have C:hyper_rel  {x} ∈
    internal_set(ConstantFunction(nat, nat)) unfolding hyper_set_def quotient_def
    using assms(2) by auto
  from e have *f*([x]) =hyper_rel
    {{⟨i, if x  i ∈ ConstantFunction(nat, nat)  i
          then (ConstantFunction(nat, f)  i)  (x  i)
          else x  i⟩ .
      i ∈ nat}}
    using internal_fun_apply_2[of ConstantFunction(nat,nat) ConstantFunction(nat,nat)
ConstantFunction(nat,f)
        x, OF A A B C] by auto
  then have *f*([x]) =hyper_rel
    {{⟨i, if x  i ∈ nat then (ConstantFunction(nat, f)  i)  (x  i) else
x  i⟩. i ∈ nat}}
    using func1_3_L2[of _ nat nat] by auto
  then have *f*([x]) =hyper_rel
    {{⟨i, (ConstantFunction(nat, f)  i)  (x  i)⟩. i ∈ nat}}
    using apply_type[OF assms(2)] by auto
  then have *f*([x]) =hyper_rel
    {{⟨i, f  (x  i)⟩. i ∈ nat}}
    using func1_3_L2[of _ nat f] by auto
  then have *f*([x]) =hyper_rel
    {{⟨i, (f O x)  i⟩. i ∈ nat}} using comp_fun_apply[OF assms(2)] by
auto
  then have *f*([x]) =hyper_rel  {f O x}
    using fun_is_set_of_pairs[OF comp_fun[OF assms(2,1)]] by auto
  then show thesis unfolding seq_class_def[OF comp_fun[OF assms(2,1)]].
qed
```

Every subset of an internal set of the natural numbers is internal

```
lemma standard_internal_set:
  assumes isInternal(A) A ⊆ ({⟨x,*x⟩. x∈nat}nat) X ⊆ A
  shows isInternal(X)
proof-
  let X={⟨x,*x⟩. x∈nat}-X
  let SX=ConstantFunction(nat,X)
  have X:X ∈Pow(nat) using vimage_iff by auto
  have s:SX:nat→Pow(nat) using func1_3_L1[OF X] by auto
```

```
  {
    fix t assume t:t∈X
    with assms(2,3) have t∈({⟨x,*x⟩. x∈nat}nat) by auto
    then obtain q where q:q∈nat t=*q using image_iff by auto
    from t q have q∈X using vimage_iff[of q {⟨x,*x⟩. x∈nat} X] by auto
    then have {n∈nat. q∈X}=nat by auto
    then have {n∈nat. q∈X}:𝔉 using ultraFilter unfolding IsUltrafilter_def
IsFilter_def by auto
    then have {n∈nat. q∈SXn}:𝔉 using func1_3_L2 by auto
    then have {n∈nat. ConstantFunction(nat,q)n∈SXn}:𝔉 using func1_3_L2
by auto
    then have hyper_rel{ConstantFunction(nat,q)}:internal_set(SX)
      unfolding internal_set_def[OF s] using func1_3_L1[OF q(1), of nat]
by auto
    then have (*q):internal_set(SX) unfolding incl_def[OF q(1)]
      seq_class_def[OF func1_3_L1[OF q(1), of nat]].
    with q(2) have t:internal_set(SX) by auto
  }
  with assms(3) have X⊆ A ∩internal_set(SX) by auto moreover
  {
    fix t assume t∈A ∩internal_set(SX)
    then have t:t∈A t∈internal_set(SX) by auto
    from t(2) obtain q where q:q:nat→nat t=hyper_rel{q}
      {n∈nat. qn:SXn}:𝔉 unfolding internal_set_def[OF s] by auto
    from q(3) have A:{n∈nat. qn:X}:𝔉 using func1_3_L2[of _ nat X] by
auto
    from t(1) assms(2) have t ∈{⟨x, *x⟩ . x ∈ nat}  nat  by auto
    then obtain s where s:s∈nat t=*s using image_iff by auto
    from q(2) s(2) have hyper_rel{q} = *s by auto
    then have ⟨q,ConstantFunction(nat, s)⟩∈hyper_rel
      unfolding incl_def[OF s(1)] seq_class_def[OF func1_3_L1[OF s(1)]]
      using eq_equiv_class[OF _ hyper_equiv func1_3_L1[OF s(1)]] by auto
    then have {n∈nat. qn = ConstantFunction(nat, s)n}:𝔉 unfolding hyper_rel_def
by auto
    then have {n∈nat. qn = s}:𝔉 using func1_3_L2[of _ nat s] by auto
    with A have {n∈nat. qn = s}∩{n∈nat. qn:X}:𝔉 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then have {n∈nat. qn = s}∩{n∈nat. qn:X} ≠0 using ultraFilter
      unfolding IsFilter_def IsUltrafilter_def by auto
    then obtain n where n∈{n∈nat. qn = s}∩{n∈nat. qn:X} by force
    then have n∈nat qn=s qn∈X by auto
    then have l:n∈nat s:X by auto
    from l(2) obtain x where x∈X x=*s using vimage_iff by auto
    with s(2) have t∈X by auto
  }
  then have A∩internal_set(SX) ⊆ X by auto
  ultimately have X= A∩internal_set(SX) by auto
  moreover have isInternal(internal_set(SX))
    unfolding isInternal_def using s by auto moreover
```

```
  note assms(1) ultimately
  show thesis using internal_inter[of A internal_set(SX)] by auto
qed
```

Every non empty internal set has a minimum element

```
theorem internal_set_has_minimum:
  assumes isInternal(A) A≠0
  shows ∃t∈A. ∀q∈A. t *≤ q
proof-
  from assms obtain S where s:S:nat→Pow(nat) A=internal_set(S)
    unfolding isInternal_def by auto
  let x={⟨i,if Si≠0 then μ x. x∈Si else 0⟩. i∈nat}
  have x:x∈nat→nat unfolding Pi_def function_def apply auto
  proof-
    fix i assume i:i∈nat Si≠0
    then obtain y where y∈Si by auto
    with s(1) i(1) have y∈Si Ord(y)
      using apply_type[of S nat λ_. Pow(nat) i] nat_into_Ord[of y] by
auto
    then have (μ x. x∈Si)∈Si using LeastI[of λq. q∈Si] by auto
    then show (μ x. x∈Si)∈nat using apply_type[OF s(1) i(1)] by auto
  qed
  {
    assume as:{n∈nat. Sn = 0}:𝔉
    {
      fix x assume x∈A
      with s(2) obtain q where x:x=[q] q:nat→nat {n∈nat. qn∈Sn}:𝔉 un-
folding internal_set_def[OF s(1)]
        using seq_class_def by auto
      from as x(3) have {n∈nat. Sn = 0}∩{n∈nat. qn∈Sn}:𝔉 using ultraFilter
        unfolding IsFilter_def IsUltrafilter_def by auto moreover
      have {n∈nat. Sn = 0}∩{n∈nat. qn∈Sn} = 0 by auto ultimately
      have 0:𝔉 by auto
      then have False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
    }
    then have False using assms(2) by auto
  }
  then have {n∈nat. Sn = 0}∉𝔉 by auto
  then have nat-{n∈nat. Sn = 0}:𝔉 using ultraFilter
    using set_ultrafilter[of {n∈nat. Sn = 0} nat 𝔉] by auto moreover
  have nat-{n∈nat. Sn = 0} = {n∈nat. Sn ≠ 0} by auto ultimately
  have {n∈nat. Sn ≠ 0}:𝔉 by auto moreover
  {
    fix n assume n∈{n∈nat. Sn ≠ 0}
    then have n:n∈nat Sn≠0 by auto
    then obtain y where y:y∈Sn by auto
    then have Ord(y) using apply_type[OF s(1) n(1)] nat_into_Ord by auto
    with y have (μ x. x:Sn):Sn using LeastI[of λq. q∈Sn y] by auto
```

```
    then have xn∈Sn using n(2) apply_equality[OF _ x] n(1) by auto
    with n(1) have n∈{n∈nat. xn∈Sn} by auto
  }
  then have {n∈nat. Sn ≠ 0} ⊆ {n∈nat. xn∈Sn} by auto moreover
  have {n∈nat. xn∈Sn} ∈Pow(nat) by auto
  ultimately have B:{n∈nat. xn∈Sn} :𝔉 using ultraFilter unfolding IsFilter_def
    IsUltrafilter_def by auto
  with x have Q:[x]∈internal_set(S) unfolding internal_set_def[OF s(1)]
    seq_class_def[OF x] by auto
  {
    fix m assume m:m∈A
    with s(2) obtain p where p:p:nat→nat m=[p] {n∈nat. pn:Sn}:𝔉
      unfolding internal_set_def[OF s(1)] using seq_class_def by auto
    from p(3) B have {n∈nat. xn∈Sn}∩{n∈nat. pn:Sn}:𝔉
      using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
moreover
    {
      fix h assume h:h∈{n∈nat. xn∈Sn}∩{n∈nat. pn:Sn}
      then have h:h∈nat xh∈Sh ph∈Sh by auto
      from h(2) have Sh≠0 by auto
      then have xx:xh=(μ x. x∈Sh) using apply_equality[OF _ x] h(1) by
auto
      from h(3) have ph∈nat using apply_type[OF s(1)] h(1) by auto
      then have Ord(ph) using nat_into_Ord by auto
      with h(3) xx have xh≤ph using Least_le[of λq. q∈Sh ph] by auto
      with h(1) have h∈{h∈nat. xh≤ph} by auto
    }
    then have {n∈nat. xn∈Sn}∩{n∈nat. pn:Sn} ⊆ {h∈nat. xh≤ph} by auto
    moreover have {h∈nat. xh≤ph}∈Pow(nat) by auto ultimately
    have {h∈nat. xh≤ph}:𝔉 using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
    then have [x]*≤[p] using less_than_seq[OF x p(1)] by auto
    with p(2) have [x]*≤m by auto
  }
  then have ∀m∈A. [x]*≤m by auto
  with Q s(2) show thesis by auto
qed
```

Every hypernatural that is not zero, is a successor

```
theorem succ_hyper_nat:
  assumes n∈*ℕ n≠*0
  shows ∃t∈*ℕ. *{⟨i,succ(i)⟩. i∈nat}⁎t=n
proof-
  let S1=ConstantFunction(nat,nat)
  let S2=ConstantFunction(nat,nat)
  let S=ConstantFunction(nat,{⟨i,succ(i)⟩. i∈nat})
  have A:S1:nat→Pow(nat) using func1_3_L1 by auto
  have B:S2:nat→Pow(nat) using func1_3_L1 by auto
  have C:S:(∏i∈nat.S1  i → S2  i)
```

```
    unfolding Pi_def function_def Sigma_def apply auto unfolding ConstantFunction_def
apply auto
    using apply_equality[of _ nat nat×{nat} nat λ_. Pow(nat)] A unfold-
ing ConstantFunction_def
    apply simp using apply_equality[of _ nat nat×{nat} nat λ_. Pow(nat)]
A unfolding ConstantFunction_def
    apply simp using apply_equality[of _ nat nat×{nat} nat λ_. Pow(nat)]
A unfolding ConstantFunction_def
    by auto
  from assms(1) obtain q where q:n=[q] q:nat→nat unfolding hyper_set_def
quotient_def
    using seq_class_def by auto
  {
    assume {n∈nat. qn =0}∈𝔉
    then have {n∈nat. qn =ConstantFunction(nat,0)n}∈𝔉
      using func1_3_L2 by auto
    then have ⟨q,ConstantFunction(nat,0)⟩∈hyper_rel
      unfolding hyper_rel_def using q(2) func1_3_L1[OF nat_0I] by auto
    then have [q] = *0 using equiv_class_eq[OF hyper_equiv]
      unfolding seq_class_def[OF q(2)] incl_def[OF nat_0I]
      seq_class_def[OF func1_3_L1[OF nat_0I]] by auto
    with q(1) assms(2) have False by auto
  }
  then have {n∈nat. qn =0}∉𝔉 by auto
  then have nat-{n∈nat. qn =0}∈𝔉 using set_ultrafilter[OF _  ultraFilter,
       of {n∈nat. qn =0}] by auto
  moreover have nat-{n∈nat. qn =0} = {n∈nat. qn ≠0} by auto
  ultimately have L:{n∈nat. qn ≠0}∈𝔉 by auto
  have suc:{⟨i, succ(i)⟩ . i ∈ nat}:nat→nat
    unfolding Pi_def function_def by auto
  let x={⟨i,pred(qi)⟩. i∈nat}
  have x:x:nat→nat unfolding Pi_def function_def apply auto
    using apply_type[OF q(2)] by auto
  then have xN:[x]∈*ℕ unfolding hyper_set_def quotient_def
    seq_class_def[OF x] by auto
  then have [x]:internal_set({⟨i,nat⟩. i∈nat})
    using internal_total_set by auto moreover
  have {⟨i,nat⟩. i∈nat} = nat×{nat} by auto
  ultimately have [x]:internal_set(nat×{nat}) by auto
  then have [x]:internal_set(S1) unfolding ConstantFunction_def.
  then have D:hyper_rel  {x} ∈ internal_set(ConstantFunction(nat, nat))
    unfolding seq_class_def[OF x] by auto
  from internal_fun_apply_2[OF A B C D]
  have Q:*{⟨i, succ(i)⟩ . i ∈ nat}*  [x] =
  hyper_rel  {{⟨i, succ(pred(qi))⟩ . i ∈ nat}} using apply_type[OF x]
func1_3_L2[of _ nat nat]
    func1_3_L2[of _ nat {⟨i, succ(i)⟩ . i ∈ nat}] apply_equality[of _
_ {⟨i, succ(i)⟩ . i ∈ nat}, OF _ suc]
    unfolding hyper_fun_def[OF suc] using apply_equality[OF _ x]
```

```
    seq_class_def[OF x]  by auto
  {
    fix i assume i∈nat qi=0
    then have pred(qi) = 0 using pred_0 by auto
    then have succ(pred(qi)) = 1 by auto
  }
  then have U:∀i∈nat. qi= 0 ⟶ succ(pred(qi)) = 1 by auto
  {
    fix i assume i:i∈nat qi≠0
    then obtain k where k:k∈nat qi = succ(k) using Nat_ZF_1_L3[OF apply_type[OF
q(2)]] i by auto
    then have pred(qi) = k using pred_succ_eq by auto
    then have succ(pred(qi)) = qi using k(2) by auto
  }
  then have V:∀i∈nat. qi≠ 0 ⟶ succ(pred(qi)) = qi by auto
  have f:{⟨i,succ(pred(qi))⟩ . i ∈ nat} = {⟨i,1⟩. i∈{p∈nat. qp = 0}} ∪
{⟨i,qi⟩. i∈{p∈nat. qp ≠ 0}}
  proof
    {
      fix j assume j∈{⟨i,succ(pred(qi))⟩ . i ∈ nat}
      then obtain i where i:i∈nat j=⟨i,succ(pred(qi))⟩ by auto
      {
        assume as:qi =0
        with U i have j=⟨i,1⟩ by auto
        then have j∈{⟨i,1⟩. i∈{p∈nat. qp = 0}} ∪ {⟨i,qi⟩. i∈{p∈nat. qp
≠ 0}}
          using i as by auto
      } moreover
      {
        assume as:qi ≠0
        with V i have j=⟨i,qi⟩ by auto
        then have j∈{⟨i,1⟩. i∈{p∈nat. qp = 0}} ∪ {⟨i,qi⟩. i∈{p∈nat. qp
≠ 0}}
          using i as by auto
      } ultimately have j∈{⟨i,1⟩. i∈{p∈nat. qp = 0}} ∪ {⟨i,qi⟩. i∈{p∈nat.
qp ≠ 0}}
        by auto
    }
    then show {⟨i,succ(pred(qi))⟩ . i ∈ nat} ⊆ {⟨i,1⟩. i∈{p∈nat. qp =
0}} ∪ {⟨i,qi⟩. i∈{p∈nat. qp ≠ 0}}
      by blast
    {
      fix j assume j:j∈{⟨i,1⟩. i∈{p∈nat. qp = 0}} ∪ {⟨i,qi⟩. i∈{p∈nat.
qp ≠ 0}}
      {
        assume j∈{⟨i,1⟩. i∈{p∈nat. qp = 0}}
        then obtain i where i∈nat qi = 0 j=⟨i,1⟩ by auto
        with U have i∈nat j=⟨i,succ(pred(qi))⟩ by auto
        then have j∈{⟨i,succ(pred(qi))⟩ . i ∈ nat} by auto
```

```
      } moreover
      {
        assume j∉{⟨i,1⟩. i∈{p∈nat. qp = 0}}
        with j have j∈{⟨i,qi⟩. i∈{p∈nat. qp ≠ 0}} by auto
        then obtain i where i∈nat qi ≠ 0 j=⟨i,qi⟩ by auto
        with V have i∈nat j=⟨i,succ(pred(qi))⟩ by auto
        then have j∈{⟨i,succ(pred(qi))⟩ . i ∈ nat} by auto
      } ultimately
      have j∈{⟨i,succ(pred(qi))⟩ . i ∈ nat} by auto
    }
    then show {⟨i,1⟩. i∈{p∈nat. qp = 0}} ∪ {⟨i,qi⟩. i∈{p∈nat. qp ≠ 0}}
⊆ {⟨i,succ(pred(qi))⟩ . i ∈ nat}
      by blast
  qed
  have ff:{⟨i,succ(pred(qi))⟩ . i ∈ nat}:nat→nat unfolding Pi_def function_def
    apply auto using apply_type[OF q(2)] by auto
  {
    fix i assume i∈{p∈nat. qp ≠ 0}
    then have i:i∈nat qi≠0 by auto
    with f have {⟨i,succ(pred(qi))⟩ . i ∈ nat}i = qi
      using apply_equality[of i qi _, OF _ ff] by auto
    with i(1) have i:{p∈nat. {⟨i,succ(pred(qi))⟩ . i ∈ nat}p = qp} by
auto
  }
  then have {p∈nat. qp ≠ 0} ⊆ {p∈nat. {⟨i,succ(pred(qi))⟩ . i ∈ nat}p
= qp} by auto
  moreover note L moreover have {p∈nat. {⟨i,succ(pred(qi))⟩ . i ∈ nat}p
= qp}:Pow(nat) by auto
  ultimately have {p∈nat. {⟨i,succ(pred(qi))⟩ . i ∈ nat}p = qp}:𝔉 us-
ing ultraFilter
    unfolding IsFilter_def IsUltrafilter_def by auto
  then have ⟨{⟨i,succ(pred(qi))⟩ . i ∈ nat},q⟩∈hyper_rel
    unfolding hyper_rel_def using ff q(2) by auto
  then have hyper_rel{{⟨i,succ(pred(qi))⟩ . i ∈ nat}} = hyper_rel{q}
    using equiv_class_eq[OF hyper_equiv] by auto
  with Q have *{⟨i, succ(i)⟩ . i ∈ nat}*  [{⟨i, Arith.pred(q  i)⟩ . i ∈
nat}] = [q]
    using seq_class_def[OF q(2)] by auto
  with q(1) have *{⟨i, succ(i)⟩ . i ∈ nat}*  [{⟨i, Arith.pred(q  i)⟩ .
i ∈ nat}] = n by auto
  with xN show thesis by auto
qed
```

The set of hypernaturals is not well ordered, since we can only prove that internal sets have minimums. Here is an example of a set without a minimum:

```
corollary not_well_ordered:
  defines NonNatural ≡ *ℕ-{⟨x,*x⟩. x∈nat}nat
  shows ∀t∈NonNatural. ∃q∈NonNatural. q *< t
```

```
proof
  fix t assume t∈NonNatural
  then have t:t∈*ℕ t∉{⟨x,*x⟩. x∈nat}nat unfolding NonNatural_def by
auto
  have f:{⟨x,*x⟩. x∈nat}:nat → *ℕ using incl_inj_nat unfolding inj_def
by auto
  with t(2) have t∉{{⟨x,*x⟩. x∈nat}n. n∈nat} using func_imagedef by auto
  then have R:t∉{*n. n∈nat} using apply_equality f by auto
  then have ∀n∈nat. t≠*n by auto
  then have t≠*0 by auto
  with t(1) obtain q where q:q∈*ℕ t = *{⟨i, succ(i)⟩ . i ∈ nat}*  q us-
ing succ_hyper_nat
    by auto
  from q(1) obtain qf where qf:[qf] = q qf:nat → nat
    unfolding hyper_set_def quotient_def using seq_class_def by auto
  have suc:{⟨i, succ(i)⟩ . i ∈ nat}:nat→nat
    unfolding Pi_def function_def by auto
  with q(2) have t:t = [{⟨i, succ(i)⟩ . i ∈ nat} O qf] using
    hyper_fun_on_nat_2[OF _ qf(2), of {⟨i,succ(i)⟩. i∈nat}] qf(1) by auto
  {
    fix n assume n∈nat
    then have A:qfn < succ(qfn) using le_refl[OF nat_into_Ord]
      apply_type qf(2) by auto
    moreover have qfn ∈ nat using apply_type qf(2) `n∈nat` by auto
    then have {⟨i, succ(i)⟩ . i ∈ nat}(qfn) = succ(qfn) using apply_equality
      suc by auto
    then have ({⟨i, succ(i)⟩ . i ∈ nat} O qf)n = succ(qfn) using
      comp_fun_apply qf(2) `n∈nat` by auto
    with A have qfn < ({⟨i, succ(i)⟩ . i ∈ nat} O qf)n by auto
  }
  then have {n∈nat. qfn < ({⟨i, succ(i)⟩ . i ∈ nat} O qf)n} = nat by
auto
  then have {n∈nat. qfn < ({⟨i, succ(i)⟩ . i ∈ nat} O qf)n} ∈𝔉 using
    ultraFilter unfolding IsUltrafilter_def using is_filter_def_split(2)
by auto
  then have [qf] *< [{⟨i, succ(i)⟩ . i ∈ nat} O qf]
    using less_seq[OF qf(2) comp_fun[OF qf(2) suc]] by auto
  with qf(1) t have q *< t by auto moreover
  {
    assume as:q∈{⟨x,*x⟩. x∈nat}nat
    then have q∈{{⟨x,*x⟩. x∈nat}n. n∈nat} using func_imagedef f by auto
    then have q∈{*n. n∈nat} using apply_equality f by auto
    then obtain n where q=*n n∈nat by auto
    then have  *{⟨i, succ(i)⟩ . i ∈ nat}*  q = *({⟨i, succ(i)⟩ . i ∈ nat}
n)
      using hyper_fun_on_nat[OF suc] by auto
    then have  *{⟨i, succ(i)⟩ . i ∈ nat}*  q = *(succ(n))
      using apply_equality[OF _ suc] `n∈nat` by auto
    then have  *{⟨i, succ(i)⟩ . i ∈ nat}*  q∈{*p. p∈nat}
```

```
      using nat_succI `n∈nat` by auto
    with q(2) have t∈{*p. p∈nat} by auto
    with R have False by auto
  }
  with q(1) have q∈NonNatural unfolding NonNatural_def by auto
  ultimately show ∃q∈NonNatural. q *< t by auto
qed
```

The natural numbers is not an internal set

```
corollary nat_not_internal:
  shows ¬isInternal({⟨x,*x⟩. x∈nat}nat)
proof
  assume isInternal({⟨x,*x⟩. x∈nat}nat)
  then have A:isInternal(*ℕ-({⟨x,*x⟩. x∈nat}nat))
    using complement_internal by auto
  {
    assume z:*ℕ-({⟨x,*x⟩. x∈nat}nat) = 0
    then have ω ∈ {⟨x,*x⟩. x∈nat}nat using omega_not_nat(2) by auto
    then obtain n where ω = *n n∈nat using image_iff by auto
    then have False using omega_not_nat(1) by auto
  }
  then have B:*ℕ-({⟨x,*x⟩. x∈nat}nat) ≠ 0 by auto
  have C:∃t∈*ℕ-({⟨x,*x⟩. x∈nat}nat). ∀q∈*ℕ-({⟨x,*x⟩. x∈nat}nat). t *≤
q
    using internal_set_has_minimum[OF A B] by auto
  then obtain t where t:t∈*ℕ-({⟨x,*x⟩. x∈nat}nat) ∀q∈*ℕ-({⟨x,*x⟩. x∈nat}nat).
t *≤ q
    by auto
  from t(1) have A:t∈*ℕ by auto
  {
    assume t=*0
    then have t∈{⟨x,*x⟩. x∈nat}nat using image_iff by auto
    with t(1) have False by auto
  }
  then have B:t ≠ *0 by auto
  then obtain q where q:q∈*ℕ *{⟨i, succ(i)⟩ . i ∈ nat}* q = t
    using succ_hyper_nat[OF A B] by auto
  have suc:{⟨i, succ(i)⟩ . i ∈ nat}:nat→nat
    unfolding Pi_def function_def by auto
  from q(1) obtain qx where qq:q=[qx] qx:nat→nat unfolding hyper_set_def
quotient_def
    using seq_class_def by auto
  from qq(1) have *{⟨i, succ(i)⟩ . i ∈ nat}* q = [{⟨i, succ(i)⟩ . i ∈
nat} O qx]
    using hyper_fun_on_nat_2[OF suc qq(2)] by auto
  then have t=[{⟨i, succ(i)⟩ . i ∈ nat} O qx] using q(2) by auto
  moreover from A obtain tx where tx:t=[tx] tx:nat→nat unfolding hyper_set_def
quotient_def
    using seq_class_def by auto
```

```
  note tx(1) ultimately have ⟨tx,{⟨i, succ(i)⟩ . i ∈ nat} O qx⟩∈hyper_rel
    using eq_equiv_class[OF _ hyper_equiv comp_fun[OF qq(2) suc]]
    unfolding seq_class_def[OF tx(2)] seq_class_def[OF comp_fun[OF qq(2)
suc]] by auto
  then have {i:nat. txi = ({⟨i, succ(i)⟩ . i ∈ nat} O qx)i}:𝔉 unfold-
ing hyper_rel_def by auto
  then have {i:nat. txi = {⟨i, succ(i)⟩ . i ∈ nat} (qxi)}:𝔉 using comp_fun_apply[OF
qq(2)]
    by auto
  then have B:{i:nat. txi = succ (qxi)}:𝔉 using apply_equality[OF _ suc]
    apply_type[OF qq(2)] by auto moreover
  have {i:nat. qxi ≤ txi}:Pow(nat) by auto moreover
  {
    fix i assume i∈{i:nat. txi = succ (qxi)}
    then have i:i∈nat txi = succ(qxi) by auto
    have qx  i ≤ qx  i Ord(qx  i)
      using nat_into_Ord[OF apply_type[OF qq(2) i(1)]] le_refl[OF nat_into_Ord[OF
apply_type[OF qq(2) i(1)]]]
      by auto
    then have (qx  i ≤ qx  i ∨ (qx  i = succ(qx  i))) ∧ Ord(qx  i) by
blast
    then have qxi ≤ succ(qxi) using le_succ_iff by auto
    then have qxi ≤ txi by (simp only: subst[OF i(2)[THEN sym], of λz.
qxi ≤ z])
    with i(1) have i∈{i:nat. qxi ≤ txi} by auto
  }
  then have {i:nat. txi = succ (qxi)} ⊆ {i:nat. qxi ≤ txi} by blast
  ultimately have A:{i:nat. qxi ≤ txi}:𝔉 using ultraFilter
    unfolding IsFilter_def IsUltrafilter_def by auto
  then have [qx] *≤ [tx] using less_than_seq[OF qq(2) tx(2)] by auto
  then have q *≤ t using qq(1) tx(1) by auto
  {
    assume q∈({⟨x,*x⟩. x∈nat}nat)
    then obtain u where u:q=*u u∈nat using image_iff by auto
    then have *{⟨i, succ(i)⟩ . i ∈ nat}*  q = *({⟨i, succ(i)⟩ . i ∈ nat}u)
      using hyper_fun_on_nat[OF suc] by auto
    then have *{⟨i, succ(i)⟩ . i ∈ nat}*  q = *(succ(u))
      using apply_equality[OF _ suc] u(2) by auto
    then have t = *(succ(u)) using q(2) by auto
    then have ⟨succ(u),t⟩∈{⟨x,*x⟩. x∈nat} using u(2) by auto
    then have t: {⟨x,*x⟩. x∈nat}nat using image_iff by auto
    with t(1) have False by auto
  }
  with q(1) have q∈*ℕ-({⟨x,*x⟩. x∈nat}nat) by auto
  with t(2) have t *≤ q by auto
  then have {i:nat. txi ≤ qxi}:𝔉 using less_than_seq[OF tx(2) qq(2)]
    using tx(1) qq(1) by auto
  with B have {i:nat. txi ≤ qxi}∩{i:nat.  tx  i = succ(qx  i)}:𝔉
    using ultraFilter unfolding IsFilter_def IsUltrafilter_def by auto
```

```
moreover
  {
    fix i assume i∈{i:nat. txi ≤ qxi}∩{i:nat.  tx  i = succ(qx  i)}
    then have i:nat txi ≤ qxi tx  i = succ(qx  i) by auto
    then have succ(qxi) ≤ qxi by auto
    then have False by auto
  }
  then have {i:nat. txi ≤ qxi}∩{i:nat.  tx  i = succ(qx  i)} = 0 by
auto
  ultimately show False using ultraFilter unfolding IsFilter_def IsUltrafilter_def
by auto
qed

lemma bij_finite:
  assumes Finite(A) Finite(B)
  shows Finite(A→B) apply (rule subset_Finite[of _ Pow(A×B)])
  prefer 2 apply (rule Finite_Pow)
proof-
  show A→B ⊆ Pow(A×B) unfolding Pi_def by auto
  from assms obtain n m where nm:A≈n B≈m n∈nat m∈nat unfolding Finite_def
by auto
  then have A*B≈n*m using prod_eqpoll_cong by auto
  then have A*B≈n⊗m
    unfolding Card_def cmult_def using nat_implies_well_ord[OF nm(4)]

      nat_implies_well_ord[OF nm(3)] well_ord_rmult [of n _ m]
      well_ord_cardinal_eqpoll[THEN eqpoll_sym, of n*m]
      eqpoll_trans[of A*B n*m |n*m|] by auto
  then have A*B≈n#*m using nat_cmult_eq_mult nm(3,4) by auto
  then show Finite(A*B) unfolding Finite_def by auto
qed

lemma hyperfinite_bijective_left_ray:
  assumes H∈Pow(*ℕ) isHyperFinite(H)
  shows ∃N∈*ℕ. ∃S∈nat→Pow(nat). ∃g∈Pi(nat, λi. Si → nat). internal_fun(g):bij(H,{i∈*ℕ.
i *< N})
proof-
  from assms(2) obtain S where S:S:nat→FinPow(nat) H=internal_set(S)
    unfolding isHyperFinite_def[OF assms(1)] by auto
  from S(1) have Y:S:nat → Pow(nat) using func1_1_L1B unfolding FinPow_def
by auto
  let N={⟨i,|Si|⟩. i∈nat}
  have n_fun:N:nat→nat unfolding Pi_def function_def using apply_type[OF
S(1)] unfolding FinPow_def
    using Finite_cardinal_in_nat by auto
  then have N:[N]:*ℕ unfolding hyper_set_def quotient_def using seq_class_def
by auto
  {
    fix g assume g:g∈Pi(nat,λi. bij(Si,|Si|))
```

```
    let f=internal_fun(g)
    have const:{⟨i, nat⟩ . i ∈ nat} = ConstantFunction(nat,nat)
      unfolding ConstantFunction_def by auto
    have S2:S:nat → Pow(nat) using S(1) unfolding FinPow_def Pi_def by
auto moreover
    have gg:g∈Pi(nat,λi. Si → nat)
      unfolding Pi_def function_def apply auto
        prefer 3 using g unfolding Pi_def function_def apply blast
       prefer 2 using g unfolding Pi_def apply blast
    proof-
      fix x assume x∈g
      with g have x∈(∑i∈nat. bij(S  i, |S  i|)) unfolding Pi_def by
auto
      then obtain i f where x:x=⟨i,f⟩ i∈nat f∈bij(S  i, |S  i|) by auto
      have f:f∈{f ∈ Pow(S  i × |Si|) .
              S  i ⊆ domain(f) ∧ (∀x y. ⟨x, y⟩ ∈ f ⟶ (∀y'. ⟨x, y'⟩
∈ f ⟶ y = y'))}
        using bij_is_fun[OF x(3)] unfolding Pi_def function_def .
      have |Si|∈nat using apply_type[OF S(1) x(2)] unfolding FinPow_def
        using Finite_cardinal_in_nat by auto
      then have |Si| ⊆ nat using OrdmemD[OF _ Ord_nat] by auto more-
over
      from f have f∈ Pow(S  i × |Si|)  by auto
      ultimately have f:  Pow(S  i × nat) by auto
      with f have f∈{f ∈ Pow(S  i × nat) .
              S  i ⊆ domain(f) ∧ (∀x y. ⟨x, y⟩ ∈ f ⟶ (∀y'. ⟨x, y'⟩
∈ f ⟶ y = y'))} by blast
      with x(1,2) show x ∈
         (∑i∈nat.
             {f ∈ Pow(S  i × nat) .
              S  i ⊆ domain(f) ∧ (∀x y. ⟨x, y⟩ ∈ f ⟶ (∀y'. ⟨x, y'⟩
∈ f ⟶ y = y'))}) by blast
    qed
    then have g2:g ∈ (∏i∈nat. S  i → ConstantFunction(nat,nat)i) us-
ing
      func1_3_L2[of _ nat nat] unfolding Pi_def by force
    have n:nat∈Pow(nat) by auto
    have fun:f:H→*ℕ using internal_fun_is_fun[OF S2 func1_3_L1[OF n]
g2]
      internal_total_set const S(2) by auto
    {
      fix t assume t:t∈range(f)
      then obtain q where ⟨q,t⟩∈f using rangeE by auto
      with fun have f:q∈H t∈*ℕ ⟨q,t⟩∈f unfolding Pi_def by auto
      from f(1,3) have fqt:fq=t using apply_equality[OF _ fun] by auto
      from f(1) S(2) obtain qx where qx:qx:nat→nat {i∈nat. qxi∈Si}∈𝔉
q=[qx]
        unfolding internal_set_def[OF S2] using seq_class_def by auto
      from f(2) obtain tx where tx:tx:nat→nat t=[tx] using seq_class_def
```

```
        unfolding hyper_set_def quotient_def by auto
      from fqt qx(3) have t = hyper_rel  {{⟨i, if qx  i ∈ S  i then g
i  (qx  i) else qx  i⟩ . i ∈ nat}} using internal_fun_apply_2[OF S2 func1_3_L1[OF
n] g2, of qx]
        f(1) S(2) unfolding seq_class_def[OF qx(1)] by auto
      with tx(2) have [tx] = hyper_rel  {{⟨i, if qx  i ∈ S  i then g
i  (qx  i) else qx  i⟩ . i ∈ nat}} by auto
      then have hyper_rel  {{⟨i, if qx  i ∈ S  i then g  i  (qx  i) else
qx  i⟩ . i ∈ nat}} = [tx] by auto
      then have ⟨{⟨i, if qx  i ∈ S  i then g  i  (qx  i) else qx  i⟩ .
i ∈ nat},tx⟩∈hyper_rel
        using eq_equiv_class[OF _ hyper_equiv, of _ tx] unfolding seq_class_def[OF
tx(1)] using tx(1) by auto
      then have {i∈nat. {⟨i, if qx  i ∈ S  i then g  i  (qx  i) else
qx  i⟩ . i ∈ nat}i = txi}:𝔉
  and ff:{⟨i, if qx  i ∈ S  i then g  i  (qx  i) else qx  i⟩ . i ∈ nat}:nat→nat
        unfolding hyper_rel_def by auto
      then have {i∈nat. {⟨i, if qx  i ∈ S  i then g  i  (qx  i) else
qx  i⟩ . i ∈ nat}i = txi}∩{i∈nat. qxi∈Si}∈𝔉
        using ultraFilter qx(2) unfolding IsFilter_def IsUltrafilter_def
by auto moreover
      {
        fix i assume i∈{i∈nat. {⟨i, if qx  i ∈ S  i then g  i  (qx  i)
else qx  i⟩ . i ∈ nat}i = txi}∩{i∈nat. qxi∈Si}
        then have i∈nat {⟨i, if qx  i ∈ S  i then g  i  (qx  i) else
qx  i⟩ . i ∈ nat}i = txi qxi∈Si by auto
        then have i:i∈nat (g  i)   (qx  i)  = txi qxi∈Si using apply_equality[OF
_ ff] by auto
        from i(1) have gi∈bij(Si,|Si|) using g apply_type by auto
        with i(3) have (gi)(qxi)∈|Si| using apply_type[OF bij_is_fun]
by auto
        with i(2) have txi∈|Si| by auto
        then have txi < |Si| unfolding lt_def by auto
        then have txi < Ni using apply_equality[of i |Si| N] n_fun i(1)
by auto
        then have i∈{i∈nat. txi < Ni } using i(1) by auto
      }
      then have {i∈nat. {⟨i, if qx  i ∈ S  i then g  i  (qx  i) else
qx  i⟩ . i ∈ nat}i = txi}∩{i∈nat. qxi∈Si} ⊆ {i∈nat. txi < Ni } by auto
moreover
      have {i∈nat. txi <Ni }:Pow(nat) by auto ultimately
      have {i∈nat. txi <Ni }:𝔉 using ultraFilter unfolding IsFilter_def
IsUltrafilter_def by auto
      then have [tx] *< [N] using less_seq[OF tx(1) n_fun] by auto
      then have t *<[N] using tx(2) by auto
      with f(2) have t:{p:*ℕ. p *<[N]} by auto
    }
    then have range(f) ⊆ {p:*ℕ. p *<[N]} by auto
    then have fun:f:H→{p:*ℕ. p *<[N]} using range_of_fun[OF fun] func1_1_L1B
```

```
by auto
    {
      fix h1 h2 assume as:h1∈H h2∈H fh1 = fh2
      from as(1,2) S(2) obtain q1 q2 where q:q1:nat→nat q2:nat→nat
{i∈nat. q1i∈Si}∈𝔉 {i∈nat. q2i∈Si}∈𝔉 h1=[q1] h2=[q2]
        unfolding internal_set_def[OF S2] using seq_class_def by auto
      let qq1={⟨i, if q1  i ∈ S  i then g  i  (q1  i) else q1  i⟩ .
          i ∈ nat}
      let qq2={⟨i, if q2  i ∈ S  i then g  i  (q2  i) else q2  i⟩ .
          i ∈ nat}
      have fh1 = hyper_rel{qq1} using internal_fun_apply(1)[OF S2 func1_3_L1[OF
n] g2 q(1,3)] q(5)
        unfolding seq_class_def[OF q(1)] by simp moreover
      have fh2 = hyper_rel{qq2} using internal_fun_apply(1)[OF S2 func1_3_L1[OF
n] g2 q(2,4)] q(6)
        unfolding seq_class_def[OF q(2)] by simp ultimately
      have eq:hyper_rel{qq1} = hyper_rel{qq2} using as(3) by auto
      moreover have s1:qq1:nat → nat using internal_fun_apply(2)[OF
S2 func1_3_L1[OF n] g2 q(1,3)] by auto
      have s2:qq2:nat → nat using internal_fun_apply(2)[OF S2 func1_3_L1[OF
n] g2 q(2,4)] by auto
      from eq have ⟨qq1, qq2⟩∈ hyper_rel using eq_equiv_class[OF _ hyper_equiv,
of
          qq1 qq2] s2 by auto
      then have {i∈nat. qq1i = qq2i}∈𝔉 unfolding hyper_rel_def by auto
      with q(3,4) have {i:nat. qq1i = qq2i}∩{i:nat. q1i:Si}∩{i:nat. q2i:Si}:𝔉
        using ultraFilter unfolding IsUltrafilter_def IsFilter_def by
auto moreover
      have {i:nat. qq1i = qq2i}∩{i:nat. q1i:Si}∩{i:nat. q2i:Si} ⊆ {i:nat.
qq1i = qq2i ∧ q1i:Si ∧ q2i:Si}
        by auto moreover
      from ultraFilter have R:⋀B C. B:𝔉 ⟹ B ⊆ C ⟹ C ⊆ nat ⟹ C:𝔉
using is_filter_def_split(5)[of 𝔉 nat]
        unfolding IsUltrafilter_def by auto
      ultimately have A:{i:nat. qq1i = qq2i ∧ q1i:Si ∧ q2i:Si}:𝔉 us-
ing ultraFilter
        by auto
      have ⋀i. i∈nat ⟹ q1i ∈ Si ⟹ qq1i = (gi)(q1i)
        using apply_equality[OF _ s1] by auto moreover
      have ⋀i. i∈nat ⟹ q2i ∈ Si ⟹ qq2i = (gi)(q2i)
        using apply_equality[OF _ s2] by auto
      ultimately have {i:nat. qq1i = qq2i ∧ q1i:Si ∧ q2i:Si} ⊆ {i:nat.
(gi)(q1i) = (gi)(q2i) ∧ q1i:Si ∧ q2i:Si}
        by auto
      with A R have F1:{i:nat. (gi)(q1i) = (gi)(q2i) ∧ q1i:Si ∧ q2i:Si}∈𝔉
by auto
      {
        fix j assume as:j∈nat (gj)(q1j) = (gj)(q2j) q1j:Sj q2j:Sj
        from g have gj∈bij(Sj,|Sj|) using apply_type as(1) by auto
```

```
        then have gj∈inj(Sj,|Sj|) unfolding bij_def by auto
        with q(1,2) as have q1j=q2j unfolding inj_def by auto
      }
      then have {i:nat. (gi)(q1i) = (gi)(q2i) ∧ q1i:Si ∧ q2i:Si} ⊆ {i:nat.
q1i = q2i} by auto
      with F1 R have {i:nat. q1i = q2i}∈𝔉 by auto
      then have ⟨q1,q2⟩∈hyper_rel unfolding hyper_rel_def using q(1,2)
by auto
      then have [q1] = [q2] using equiv_class_eq[OF hyper_equiv] seq_class_def
        q(1,2) by auto
      with q(5,6) have h1=h2 by auto
    }
    with fun have f∈inj(H,{p∈*ℕ. p *< [N]}) unfolding inj_def by auto
    {
      fix p assume as:p∈{p∈*ℕ. p *< [N]}
      then have p:p∈*ℕ p *<[N] by auto
      from p(1) obtain pf where pf:pf:nat→nat p =[pf] unfolding hyper_set_def
        quotient_def seq_class_def by auto
      from pf p(2) n_fun have {n∈nat. pfn < Nn}∈𝔉 using less_seq by
auto
      moreover have ⋀n. n∈nat ⟹ Nn = |Sn| using apply_equality[OF
_ n_fun] by auto
      ultimately have A:{n∈nat. pfn < |Sn|}∈𝔉 by auto
      let h={⟨i,if pfi∈|Si| then converse(gi)(pfi) else 0⟩. i∈nat}
      {
        fix i assume as:i∈nat pfi ∉ |Si|
        from as(2) have (if pfi∈|Si| then converse(gi)(pfi) else 0) =
0 by auto
        then have ⟨i,if pfi∈|Si| then converse(gi)(pfi) else 0⟩∈nat×nat
using as(1) by auto
      } moreover
      {
        fix i assume as:i∈nat pfi∈|Si|
        then have (if pfi∈|Si| then converse(gi)(pfi) else 0) = converse(gi)(pfi)
by auto
        moreover note as(1) moreover
        from g have gi∈bij(Si,|Si|) using apply_type as(1) by auto
        then have converse(gi)∈bij( |Si|, Si) using bij_converse_bij
by auto
        with as(2) have converse(gi)(pfi)∈Si using apply_type[OF bij_is_fun]
by auto
        moreover have Si ⊆ nat using apply_type[OF S2 as(1)] by auto
        ultimately have (if pfi∈|Si| then converse(gi)(pfi) else 0)∈nat
by auto
      }
      ultimately have h ⊆ nat×nat by auto
      then have h_fun:h:nat → nat unfolding Pi_def function_def by auto
      {
        fix m assume m:m∈nat pfm <|Sm|
```

```
        with m(2) have pfm∈|Sm| unfolding lt_def by auto
        then have (if pfm∈|Sm| then converse(gm)(pfm) else 0) = converse(gm)(pfm)
by auto
        then have hm = converse(gm)(pfm) using apply_equality m(1) h_fun
by auto
        moreover from m(1) g have gm∈bij(Sm,|Sm|) using apply_type by
auto
        then have converse(gm)∈bij( |Sm|, Sm) using bij_converse_bij
by auto
        then have converse(gm)(pfm)∈Sm using apply_type[OF bij_is_fun]
m(2) unfolding lt_def by auto
        ultimately have hm∈Sm by auto
      }
      then have B:{m∈nat. pfm <|Sm| } ⊆ {m∈nat. hm∈Sm} by auto
      let p={⟨i, if h  i ∈ S  i then g  i  (h  i) else h  i⟩ . i ∈ nat}
      have 𝔉{is a filter on}nat using ultraFilter unfolding IsUltrafilter_def
by auto
      then have ⋀B C. B∈𝔉 ⟹ C∈Pow(nat) ⟹ B ⊆ C ⟹ C ∈ 𝔉 using
is_filter_def_split(5) by auto
      with A have CC:⋀C. C ⊆ nat ⟹ {m∈nat. pfm <|Sm| } ⊆ C ⟹ C∈𝔉
by auto
      then have {m∈nat. hm∈Sm} ⊆ nat ⟹ {m∈nat. pfm <|Sm| } ⊆ {m∈nat.
hm∈Sm} ⟹ {m∈nat. hm∈Sm}∈𝔉 by auto
      with B have C:{m∈nat. hm∈Sm}∈ 𝔉 by auto
      with h_fun have T:[h]∈H using S(2) unfolding internal_set_def[OF
S2] using seq_class_def by auto
      have nat∈Pow(nat) by auto
      then have Q:ConstantFunction(nat,nat):nat → Pow(nat) using func1_3_L1[of
nat Pow(nat)] by auto
      have im:f[h] = hyper_rel{p}
        unfolding seq_class_def[OF h_fun] using internal_fun_apply(1)[OF
S2 Q g2 h_fun C] by auto
      have p:nat→nat using internal_fun_apply(2)[OF S2 Q g2 h_fun C]
by auto
      {
        fix i assume n:i∈nat and as:pfi∈|Si|
        then have p:pi = (if h  i ∈ S  i then g  i  (h  i) else h  i)
using apply_equality
          `p:nat→nat` by auto
        have h:hi = (if pfi∈|Si| then converse(gi)(pfi) else 0) using
h_fun apply_equality n by auto
        with as have A:hi = converse(gi)(pfi) by auto moreover
        from n g have B:gi∈bij(Si,|Si|) using apply_type by auto
        then have converse(gi)∈bij(|Si|, Si) using bij_converse_bij by
auto
        then have converse(gi)(pfi)∈Si using apply_type[OF bij_is_fun]
n as unfolding lt_def by auto
        ultimately have hi∈Si by auto
        with p have pi = g  i  (h  i) by auto
```

```
        then have pi = g  i (converse(gi)(pfi)) using subst[OF A, of
λq. pi = giq]
          by auto
        then have pi = pfi using right_inverse_bij[OF B as] by auto
      }
      then have {i∈nat. pfi∈|Si|} ⊆ {i∈nat. pi = pfi} by auto
      then have {i∈nat. pfi<|Si|} ⊆ {i∈nat. pi = pfi} unfolding lt_def
by auto
      moreover from CC have {i∈nat. pi = pfi} ⊆ nat ⟹ {m∈nat. pfm
<|Sm| } ⊆ {i∈nat. pi = pfi}⟹ {i∈nat. pi = pfi}∈𝔉
        by auto
      ultimately have {i∈nat. pi = pfi}∈𝔉 by auto
      then have ⟨p,pf⟩∈hyper_rel unfolding hyper_rel_def using `p:nat→nat`
pf(1) by auto
      then have [p] = [pf] using `p:nat→nat` pf(1) using equiv_class_eq[OF
hyper_equiv]
        seq_class_def by auto
      with pf(2) have [p] = p by auto
      with im have f[h] = p using seq_class_def `p:nat→nat` by auto
      with T have ∃h∈H. fh = p by auto
    }
    then have f∈surj(H,{p∈*ℕ. p *< [N]})
      unfolding surj_def using fun by auto
    with `f∈inj(H,{p∈*ℕ. p *< [N]})` have f∈bij(H,{p∈*ℕ. p *< [N]})
      unfolding bij_def by auto
    then have ∃SS∈nat→Pow(nat). ∃g∈Pi(nat, λi. SSi → nat). internal_fun(g)∈bij(H,{p∈*ℕ
p *< [N]}) using gg Y by auto
  }
  then have R1:⋀g. g ∈(∏i∈nat. bij(S  i, |S  i|)) ⟹ ∃SS∈nat→Pow(nat).
∃Sa∈Pi(nat, λi. SSi → nat). internal_fun(Sa) ∈ bij(H, {p ∈ *ℕ. p *<
[N]}) by auto
  let g={⟨i,{⟨q,|{p∈Si. p < q}|⟩. q∈Si}⟩. i∈nat}
  {
    fix i assume as1:i∈nat
    let u={⟨q,|{p∈Si. p < q}|⟩. q∈Si}
    {
      fix q assume as2:q∈Si
      have sub:{p∈Si. p < q} ⊆ Si by auto
      from S(1) as1 have C:Si∈FinPow(nat) using apply_type by auto
      with sub have {p∈Si. p < q}∈FinPow(nat) using subset_finpow by
auto
      moreover from as2 C have q∈nat unfolding FinPow_def by auto more-
over
      {
        assume q∈{p∈Si. p < q}
        then have q<q by auto
        then have False unfolding lt_def using mem_irrefl by auto
      }
      ultimately have {p∈Si. p < q}∈FinPow(nat) q∈nat-{p∈Si. p < q} by
```

```
auto
      then have |{p∈Si. p<q}∪{q}| = succ(|{p∈Si. p<q}|) using card_fin_add_one
by auto
      moreover have {p∈Si. p<q}∪{q} ⊆ Si using as2 sub by auto
      then have l:{p∈Si. p<q}∪{q} ≲ Si using subset_imp_lepoll by auto
      have Finite(Si) using C unfolding FinPow_def by auto
      then obtain r where well_ord(Si,r) using Finite_imp_well_ord by
auto
      with l have |{p∈Si. p<q}∪{q}| ≤ |Si| using well_ord_lepoll_imp_cardinal_le
by auto
      ultimately have succ(|{p∈Si. p<q}|) ≤ |Si| using subst[of _ succ(|{p∈Si.
p<q}|)
        λq. q≤|Si|] by auto moreover
      have |{p∈Si. p<q}| < succ(|{p∈Si. p<q}|) using le_refl[OF Ord_cardinal]
by auto
      ultimately have |{p∈Si. p<q}| < |Si| using lt_trans2[of |{p∈Si.
p<q}| succ(|{p∈Si. p<q}|)] by auto
      then have |{p∈Si. p<q}| ∈ |Si| unfolding lt_def by auto
    }
    then have u_fun:u:Si → |Si| unfolding Pi_def function_def by auto
    {
      fix q1 q2 assume as:q1∈Si q2∈Si uq1 = uq2
      then have M:|{p∈Si. p<q1}| = |{p∈Si. p<q2}| using apply_equality
u_fun by auto
      from S(1) as1 have Q:Si∈FinPow(nat) using apply_type by auto
      with as(1,2) have nat:q1∈nat q2∈nat unfolding FinPow_def by auto
      {
        assume eq:q1 < q2
        with as(1) have A:q1∈{p∈Si. p<q2} by auto
        have B:q1∉{p∈Si. p<q1} using mem_irrefl unfolding lt_def by auto
        {
          assume {p∈Si. p<q1}={p∈Si. p<q2}
          with A B have False by auto
        }
        then have C:{p∈Si. p<q1}≠{p∈Si. p<q2} by auto
        {
          fix p assume p∈{p∈Si. p<q1}
          then have p∈Si p<q1 by auto
          then have p∈Si p<q2 using lt_trans eq by auto
          then have p∈{p∈Si. p<q2} by auto
        }
        then have D:{p∈Si. p<q1}⊆{p∈Si. p<q2} by auto
        from Q have {p∈Si. p<q2}∈FinPow(nat) using subset_finpow by
blast
        then have E:Finite({p∈Si. p<q2}) unfolding FinPow_def by auto
        from C D E have |{p∈Si. p<q1}|<|{p∈Si. p<q2}| using strict_subset_smaller
by auto
        with M have False using lt_irrefl by auto
      }
```

```
      then have E1:q2≤q1 using not_lt_iff_le[OF nat_into_Ord[OF nat(1)]
nat_into_Ord[OF nat(2)]] by blast
      {
        assume eq:q2 < q1
        with as(2) have A:q2∈{p∈Si. p<q1} by auto
        have B:q2∉{p∈Si. p<q2} using mem_irrefl unfolding lt_def by auto
        {
          assume {p∈Si. p<q1}={p∈Si. p<q2}
          with A B have False by auto
        }
        then have C:{p∈Si. p<q2}≠{p∈Si. p<q1} by auto
        {
          fix p assume p∈{p∈Si. p<q2}
          then have p∈Si p<q2 by auto
          then have p∈Si p<q1 using lt_trans eq by auto
          then have p∈{p∈Si. p<q1} by auto
        }
        then have D:{p∈Si. p<q2}⊆{p∈Si. p<q1} by auto
        from Q have {p∈Si. p<q1}∈FinPow(nat) using subset_finpow by
blast
        then have E:Finite({p∈Si. p<q1}) unfolding FinPow_def by auto
        from C D E have |{p∈Si. p<q2}|<|{p∈Si. p<q1}| using strict_subset_smaller
by auto
        with M have False using lt_irrefl by auto
      }
      then have E2:q1≤q2 using not_lt_iff_le[OF nat_into_Ord[OF nat(2)]
nat_into_Ord[OF nat(1)]] by blast
      with E1 have q1=q2 using le_anti_sym by auto
    }
    with u_fun have u∈inj(Si,|Si|) unfolding inj_def by auto moreover
    from S(1) as1 have C:Si∈FinPow(nat) using apply_type by auto
    then have S:Si ⊆ nat Finite(Si) unfolding FinPow_def by auto
    then have Finite(|Si|) using lepoll_Ord_imp_eqpoll[OF subset_imp_lepoll
Ord_nat] eqpoll_imp_Finite_iff by auto
    moreover note S(1)
    ultimately have u∈bij(Si,|Si|) using inj_function_same_card_bij[OF
_ lepoll_Ord_imp_eqpoll[THEN eqpoll_sym, OF subset_imp_lepoll Ord_nat]]
by auto
  }
  then have g∈Pi(nat,λi. bij(Si,|Si|)) unfolding Pi_def function_def
by auto
  with R1 have ∃SS∈nat→Pow(nat). ∃g∈Pi(nat, λi. SSi → nat). internal_fun(g)
∈ bij(H, {p ∈ *ℕ. p *< [N]}) by auto
  then show thesis using N by auto
qed

lemma internal_inj_fun_nat:
  assumes S1: nat → Pow(nat) S2 ∈ nat → Pow(nat) S ∈ (∏i∈nat. S1
i → S2  i)
```

```
    and internal_fun(S)∈inj(internal_set(S1), internal_set(S2))
  shows {n∈nat. Sn∈inj(S1n,S2n)}∈𝔉 apply (rule internal_inj_fun)
  using assms(1) apply simp using assms(2) apply simp using assms(3) ap-
ply simp using assms(4) apply simp
proof-
  show nat → nat ≠0 by auto
  let q1 = λn. μ q1. ∃q2. (Sn)q1 = (Sn)q2 ∧ q1≠q2 ∧ q1∈S1n ∧ q2∈S1n
  let q2 = λn. μ q2. (Sn)q1(n) = (Sn)q2 ∧ q1(n)≠q2 ∧ q2∈S1n
  {
    fix n assume as:n∈nat Sn∉inj(S1n,S2n)
    from as obtain q1 q2 where q:(Sn)q1 = (Sn)q2 q1≠q2 q1∈S1n q2∈S1n
unfolding inj_def using apply_type[OF assms(3)] by auto
    from q(1,2) have ∃q2. (Sn)q1(n) = (Sn)q2 ∧ q1(n)≠q2 ∧ q1(n)∈S1n
∧ q2∈S1n using LeastI[of λt. ∃q2. (Sn)t = (Sn)q2 ∧ t≠q2 ∧ t∈S1n ∧ q2∈S1n]
      nat_into_Ord[of q1] apply_type[OF assms(1) as(1)] q(3,4) by blast
    then obtain q3 where qq:(Sn)q1(n) = (Sn)q3 ∧ q1(n)≠q3 q1(n)∈S1n
q3∈S1n by auto
    then have (Sn)q1(n) = (Sn)q2(n) q1(n)≠q2(n) q2(n)∈S1n using LeastI[of
λt. (Sn)q1(n) = (Sn)t ∧ q1(n)≠t ∧ t∈S1n]
      nat_into_Ord[of q3] apply_type[OF assms(1) as(1)] by auto
    then have ⟨q1(n),q2(n)⟩∈{⟨q1,q2⟩ ∈ S1  n × S1n . S  n  q1 = S  n
q2 ∧ q1 ≠ q2}
      using apply_type[OF assms(1) as(1)] qq(2) by auto
  }
  then have {⟨n,⟨q1(n),q2(n)⟩⟩. n∈{n ∈ nat . S  n ∉ inj(S1  n, S2n)}}
∈ Pi({n ∈ nat . S  n ∉ inj(S1  n, S2n)},λn. {⟨q1,q2⟩ ∈ S1  n × S1n .
S  n  q1 = S  n  q2 ∧ q1 ≠ q2})
    unfolding Pi_def function_def Sigma_def by auto
  then show (∏n∈{n ∈ nat . S  n ∉ inj(S1  n, S2n)}. {⟨q1,q2⟩ ∈ S1
n × S1n . S  n  q1 = S  n  q2 ∧ q1 ≠ q2}) ≠ 0
    by auto
qed

lemma hyperfinite_bijective_left_ray_rev:
  assumes ∃N∈*ℕ. ∃S∈nat→Pow(nat). ∃g∈Pi(nat, λi. Si → nat). internal_fun(g):bij(H,{i∈*
i *< N})
  shows isHyperFinite(H)
proof-
  from assms obtain N S g where NSG:N∈*ℕ S:nat→Pow(nat) g∈Pi(nat,λi.
Si → nat) internal_fun(g)∈bij(H,{i∈*ℕ. i *< N}) by auto
  have ⋀i. i∈nat ⟹ ConstantFunction(nat,nat)i = nat using func1_3_L2
by auto
  with NSG(3) have gg:g∈Pi(nat,λi. Si → ConstantFunction(nat,nat)i)
unfolding Pi_def by force
  then have internal:internal_fun(g)∈internal_set(S) → internal_set(ConstantFunction(nat,n
    using internal_fun_is_fun(1)[OF NSG(2) func1_3_L1[of nat], of g] by
auto
  then have H:H=internal_set(S) using bij_is_fun[OF NSG(4)] unfolding
Pi_def domain_def by blast
```

```
  then have sub:H∈Pow(*ℕ) using internal_subset NSG(2) by auto
  let S={⟨n,if Finite(Sn) then Sn else 0⟩. n∈nat}
  from internal NSG(4) have internal_fun(g)∈inj(internal_set(S), internal_set(ConstantFunct
    unfolding bij_def inj_def using H by auto
  then have {n∈nat. gn∈inj(Sn,ConstantFunction(nat,nat)n)}∈𝔉
    using internal_inj_fun_nat[of S ConstantFunction(nat,nat) g] using
gg NSG(2)
    func1_3_L1[of nat Pow(nat)] by auto
  then have {n∈nat. gn∈inj(Sn,nat)}∈𝔉 using func1_3_L2 by auto
  from NSG(1) obtain n where n:n:nat→nat hyper_rel{n}= N unfolding hyper_set_def
quotient_def by auto
  let N={⟨q,{m∈nat. m < nq}⟩. q∈nat}
  {
    fix q assume q∈nat
    then have q:nq∈nat using apply_type n(1) by auto
    then have Ord(nq) nq ⊆ nat using nat_into_Ord lt_nat_in_nat[of _
nq] unfolding lt_def by auto
    then have Q:nq = {m∈nat. m < nq} unfolding lt_def by auto
    then have Finite({m∈nat. m < nq}) using nat_into_Finite q by auto
    with Q have {m∈nat. m < nq}∈FinPow(nat) unfolding FinPow_def by blast
  }
  then have NF:N:nat → FinPow(nat) and N:N:nat → Pow(nat) unfolding
Pi_def function_def by auto
  {
    fix t assume t:t∈internal_set(N)
    then obtain q where q:q:nat→nat t=[q] {n∈nat. qn∈Nn}∈𝔉
      unfolding internal_set_def[OF N] seq_class_def by auto
    from q(1,2) have t∈*ℕ unfolding hyper_set_def seq_class_def[OF q(1)]
by auto
    from q(3) have A:{s∈nat. qs∈{m∈nat. m < ns}}∈𝔉 using apply_equality[OF
_ N] by auto
    have {s∈nat. qs∈{m∈nat. m < ns}} ⊆ {s∈nat. qs < ns} by auto
    moreover have ⋀C. C⊆nat ⟹ {s∈nat. qs∈{m∈nat. m < ns}} ⊆ C ⟹
C∈𝔉 using ultraFilter
      unfolding IsUltrafilter_def using is_filter_def_split(5) A by auto
    then have {s∈nat. qs < ns}⊆nat ⟹ {s∈nat. qs∈{m∈nat. m < ns}} ⊆
{s∈nat. qs < ns} ⟹ {s∈nat. qs < ns}∈𝔉
      by auto
    ultimately have {s∈nat. qs < ns}∈𝔉 by auto
    then have [q] *< [n] using less_seq[OF q(1) n(1)] by auto
    with q(2) n(2) have t *< N unfolding seq_class_def[OF q(1)] seq_class_def[OF
n(1)] by auto
    with `t∈*ℕ` have t∈{p∈*ℕ. p *< N} by auto
  }
  then have internal_set(N) ⊆ {p∈*ℕ. p *< N} by auto moreover
  {
    fix t assume t∈{p∈*ℕ. p *< N}
    then have t:t∈*ℕ t *< [n] using n seq_class_def by auto
    from t(1) obtain x where x:x:nat→nat t=[x] unfolding hyper_set_def
```

```
quotient_def using seq_class_def by auto
    with t(2) have [x] *< [n] by auto
    then have {s∈nat. xs < ns} ∈𝔉 using less_seq[OF x(1) n(1)] by auto
    moreover have ⋀s. s∈nat ⟹ xs∈nat using apply_type x(1) by auto
    ultimately have {s∈nat. xs∈{m∈nat. m<ns}}∈𝔉 by auto
    then have {s∈nat. xs∈Ns}∈𝔉 using apply_equality[OF _ N] by auto
    then have [x]∈internal_set(N) unfolding internal_set_def[OF N] us-
ing x(1) seq_class_def by auto
    with x(2) have t∈internal_set(N) by auto
  }
  ultimately have NN:internal_set(N) = {p∈*ℕ. p *< N} by auto
  let Q={⟨i,(gi)(Si)⟩. i∈nat}
  {
    fix q assume q∈nat
    then have gq:Sq → nat using NSG(3) apply_type by auto
    then have (gq)(Sq) ⊆ nat using func1_1_L6 by auto
  }
  then have Q:Q:nat → Pow(nat)unfolding Pi_def function_def by auto
  {
    fix t assume t∈internal_set(Q)
    with Q obtain x where x:x:nat→nat t=[x] {q∈nat. xq∈Qq}∈𝔉
      unfolding internal_set_def[OF Q] seq_class_def by auto
    from x(3) have {q∈nat. xq∈(gq)(Sq)}∈𝔉 using apply_equality[OF _
Q] by auto
    let u=λi. μ u. u∈Si ∧ (gi)u = xi
    let y={⟨i,if (xi∈(gi)(Si)) then u(i) else 0⟩. i∈nat}
    {
      fix i assume i:i∈nat
      from i have g:gi:Si→ nat using NSG(3) apply_type by auto
      {
        assume as:xi∈(gi)(Si)
        then obtain u where u∈Si (gi)u = xi using func_imagedef[OF g]
by auto
        moreover from `u∈Si` have u∈nat using apply_type[OF NSG(2)]
i by auto
        then have Ord(u) using nat_into_Ord by auto
        ultimately have u:u(i)∈Si (gi)u(i) = xi
          using LeastI[where P=λu. u∈Si ∧ (gi)u = xi] by auto
        from u(1) have u(i)∈nat using apply_type[OF NSG(2)] i by auto
      } moreover
      {
        assume as:xi∉(gi)(Si)
        have 0∈nat by auto
      }
      ultimately have (if (xi∈(gi)(Si)) then u(i) else 0)∈nat by auto
    }
    then have y ⊆ nat×nat by auto
    then have y:y:nat → nat unfolding Pi_def function_def by auto
    {
```

```
      fix q assume q∈{q∈nat. xq∈(gq)(Sq)}
      then have q:q∈nat xq∈(gq)(Sq) by auto
      with y have yu:yq = u(q) using apply_equality by auto
      from q(1) have g:gq:Sq→ nat using NSG(3) apply_type by auto
      from q(2) obtain u where u∈Sq (gq)u = xq using func_imagedef[OF
g] by auto
      moreover from `u∈Sq` have u∈nat using apply_type[OF NSG(2)] q(1)
by auto
      then have Ord(u) using nat_into_Ord by auto
      ultimately have u:u(q)∈Sq (gq)u(q) = xq
        using LeastI[where P=λu. u∈Sq ∧ (gq)u = xq] by auto
      from u yu have yq∈Sq (gq)u(q) = xq by auto
    }
    then have D1:{q∈nat. xq∈(gq)(Sq)} ⊆ {q∈nat. yq∈Sq} and
    D2:{q∈nat. xq∈(gq)(Sq)} ⊆ {q∈nat. (gq)u(q) = xq} by auto
    note D1 moreover note `{q∈nat. xq∈(gq)(Sq)}∈𝔉` moreover
    from ultraFilter have ⋀B C. C⊆ nat ⟹ B⊆ C ⟹ B∈𝔉 ⟹ C∈𝔉
      unfolding IsUltrafilter_def using is_filter_def_split(5) by auto
    then have {q∈nat. yq∈Sq}⊆ nat ⟹ {q∈nat. xq∈(gq)(Sq)}⊆ {q∈nat.
yq∈Sq} ⟹ {q∈nat. xq∈(gq)(Sq)}∈𝔉 ⟹ {q∈nat. yq∈Sq}∈𝔉
      by auto
    ultimately have Z:{q∈nat. yq∈Sq}∈𝔉 by auto
    then have y∈{f∈nat→nat. {q∈nat. fq∈Sq}∈𝔉} using y by auto
    then have yS:[y]∈internal_set(S) unfolding internal_set_def[OF NSG(2)]
seq_class_def[OF y] by auto
    with H have [y]∈H by auto
    then have internal_fun(g)[y]∈{i∈*ℕ. i *< N} using apply_type[OF
bij_is_fun[OF NSG(4)]] by auto
    with NN have P:internal_fun(g)[y]∈internal_set(N) by auto
    let z={⟨i, if y  i ∈ S  i
                 then g  i  (y  i)
                 else y  i⟩ .
                 i ∈ nat}
    from yS have z:internal_fun(g)[y] = hyper_rel  {z} z:nat→nat us-
ing internal_fun_apply_2[OF NSG(2) func1_3_L1[OF _] gg] unfolding seq_class_def[OF
y]
      by auto
    {
      fix q assume q∈{q∈nat. yq∈Sq}∩{q∈nat. xq∈(gq)(Sq)}
      then have q:q∈nat yq∈Sq xq∈(gq)(Sq) by auto
      from q(1) have zq = (if y  q ∈ S  q
                 then g  q  (y  q)
                 else y  q) using apply_equality z(2) by auto
      with q(2) have zq = gq(yq) by auto
      with q(3) have zq = (gq)u(q) using apply_equality q(1) y by auto
      moreover from q(1,3) D2 have (gq)u(q) = xq by auto
      ultimately have zq = xq by auto
    }
    then have {q∈nat. yq∈Sq}∩{q∈nat. xq∈(gq)(Sq)} ⊆ {q∈nat. zq = xq}
```

```
by auto
    moreover have {q∈nat. yq∈Sq}∩{q∈nat. xq∈(gq)(Sq)}∈𝔉 using
      `{q∈nat. xq∈(gq)(Sq)}∈𝔉` Z ultraFilter unfolding IsUltrafilter_def

      using is_filter_def_split(4) by auto
    moreover have ⋀C B. C⊆nat ⟹ B⊆C ⟹ B∈𝔉 ⟹ C∈𝔉 using ultraFilter
unfolding IsUltrafilter_def
      using is_filter_def_split(5) by auto
    ultimately have {q∈nat. zq = xq}⊆nat ⟹ {q∈nat. zq = xq}∈𝔉 by auto
    then have {q∈nat. zq = xq}∈𝔉 by auto
    then have ⟨z,x⟩∈hyper_rel unfolding hyper_rel_def using z x(1) by
auto
    then have [z] = [x] using equiv_class_eq[OF hyper_equiv] seq_class_def
z x(1) by auto
    with x(2) have [z] = t by auto
    with z(1) have internal_fun(g)[y] = t using seq_class_def z(2) by
auto
    with P have t∈internal_set(N) by auto
  }
  then have internal_set(Q) ⊆ internal_set(N) by auto
  moreover
  have id(nat)∈nat→nat using id_bij unfolding bij_def inj_def by auto
  then have nat→nat ≠0 by auto moreover note Q N moreover
  let qq={⟨j,μ p. p∈Qj-Nj⟩. j∈{i∈nat. ¬(Qi ⊆ Ni)}}
  {
    fix j assume j∈{i∈nat. ¬(Qi ⊆ Ni)}
    then have j:j∈nat ¬(Qj ⊆ Nj) by auto
    from j(1) have g:gj:Sj→ nat using NSG(3) apply_type by auto
    from j(2) obtain u where u:u∈Qj - Nj by auto
    then have u∈Qj by auto
    then have u∈(gj)(Sj) using j(1) Q apply_equality by auto
    then obtain v where v∈Sj (gj)v = u using func_imagedef[OF g] by
auto
    with g have u∈nat using apply_type by auto
    then have Ord(u) using nat_into_Ord by auto
    with u have (μ p. p∈Qj -Nj)∈Qj -Nj using LeastI[where P=λp. p∈Qj
-Nj] by auto
  }
  then have qq∈Pi({i∈nat. ¬(Qi ⊆ Ni)},λj. Qj-Nj) unfolding Pi_def function_def
    by auto ultimately
  have {n∈nat. Qn ⊆ Nn}∈𝔉 using internal_sub_rev by auto
  let S={⟨i,if Finite(Si) then Si else 0⟩. i∈nat}
  {
    fix q assume q∈nat
    {
      assume Finite(Sq)
      then have A:(if Finite(Sq) then Sq else 0) = Sq by auto
      with `Finite(Sq)` have Finite(if Finite(Sq) then Sq else 0) by
auto
```

```
      moreover from A have (if Finite(Sq) then Sq else 0) ⊆ nat using apply_type[OF NSG(2)]
        `q∈nat` by auto
      ultimately have (if Finite(Sq) then Sq else 0)∈FinPow(nat) unfolding FinPow_def by auto
    }
    moreover
    {
      assume ¬Finite(Sq)
      then have A:(if Finite(Sq) then Sq else 0) = 0 by auto
      then have (if Finite(Sq) then Sq else 0)∈FinPow(nat) unfolding FinPow_def by auto
    }
    ultimately have (if Finite(Sq) then Sq else 0)∈FinPow(nat) by auto
  }
  then have SS:S∈nat → FinPow(nat) and S:S∈nat → Pow(nat) unfolding Pi_def function_def FinPow_def by auto
  {
    fix q assume q∈{n∈nat. Qn ⊆ Nn}∩{n∈nat. gn∈inj(Sn,nat)}
    then have q:q∈nat Qq ⊆ Nq gq∈inj(Sq,nat) by auto
    from NF q(1) have Nq∈FinPow(nat) using apply_type by auto
    then have Finite(Nq) unfolding FinPow_def by auto
    with q(2) have Finite(Qq) using subset_Finite by auto
    then have Finite((gq)(Sq)) using apply_equality Q q(1) by auto
    moreover from q(3) have gq∈bij(Sq,range(gq)) using inj_bij_range by auto
    then have Sq ≈ range(gq) unfolding eqpoll_def by auto
    then have Sq ≈ (gq)(Sq) using range_image_domain inj_is_fun[OF q(3)] by auto
    ultimately have Finite(Sq) using eqpoll_imp_Finite_iff by auto
    then have (if Finite(Sq) then Sq else 0) = Sq by auto
    then have Sq = Sq using apply_equality q(1) SS by auto
  }
  then have {n∈nat. Qn ⊆ Nn}∩{n∈nat. gn∈inj(Sn,nat)} ⊆ {q∈nat. Sq = Sq} by auto
  moreover have {q∈nat. Sq = Sq} ⊆ nat by auto
  moreover have {n∈nat. Qn ⊆ Nn}∩{n∈nat. gn∈inj(Sn,nat)}:𝔉 using
    `{n∈nat. Qn ⊆ Nn}:𝔉` `{n∈nat. gn∈inj(Sn,nat)}∈𝔉` ultraFilter
    unfolding IsUltrafilter_def using is_filter_def_split(4) by auto
  ultimately have {q∈nat. Sq = Sq}∈𝔉 using ultraFilter
    unfolding IsUltrafilter_def using is_filter_def_split(5) by auto
  then have internal_set(S) = internal_set(S) using internal_eq S NSG(2) by auto
  with H have H = internal_set(S) by auto
  with SS have ∃S∈nat → FinPow(nat). H= internal_set(S) by auto
  then show thesis unfolding isHyperFinite_def[OF sub].
qed

end
```

end

93 More on uniform spaces

```
theory UniformSpace_ZF_1 imports func_ZF_1 UniformSpace_ZF Topology_ZF_2
begin
```

This theory defines the maps to study in uniform spaces and proves their basic properties.

93.1 Uniformly continuous functions

Just as the the most general setting for continuity of functions is that of topological spaces, uniform spaces are the most general setting for the study of uniform continuity.

A map between 2 uniformities is uniformly continuous if it preserves the entourages:

```
definition
  IsUniformlyCont (_ {is uniformly continuous between} _ {and} _ 90) where
  f:X→Y ⟹ Φ {is a uniformity on} X  ⟹ Γ {is a uniformity on} Y ⟹

  f {is uniformly continuous between} Φ {and} Γ ≡ ∀V∈Γ. (ProdFunction(f,f)-V)
∈ Φ
```

Any uniformly continuous function is continuous when considering the topologies on the uniformities.

```
lemma uniformly_cont_is_cont:
  assumes f:X→Y Φ {is a uniformity on} X  Γ {is a uniformity on} Y
    f {is uniformly continuous between} Φ {and} Γ
  shows IsContinuous(UniformTopology(Φ,X),UniformTopology(Γ,Y),f)
proof -
  {  fix U assume op: U ∈ UniformTopology(Γ,Y)
    have f-(U) ∈ UniformTopology(Φ,X)
    proof -
      from assms(1) have f-(U) ⊆ X using func1_1_L3 by simp
      moreover
      { fix x xa assume as:⟨x,xa⟩ ∈ f xa ∈ U
        with assms(1) have x:x∈X unfolding Pi_def by auto
        from as(2) op have U:U ∈ {⟨t,{V{t}.V∈Γ}⟩.t∈Y}(xa) using uniftop_def_alt
by auto
        from as(1) assms(1) have xa:xa ∈ Y unfolding Pi_def by auto
        have {⟨t,{V{t}.V∈Γ}⟩.t∈Y} ∈ Pi(Y,%t. {{V{t}.V∈Γ}}) unfolding
Pi_def function_def
          by auto
        with U xa have U ∈{V{xa}.V∈Γ} using apply_equality by auto
        then obtain V where V:U = V{xa} V∈Γ by auto
```

```
        with assms have ent:(ProdFunction(f,f)-(V))∈Φ using IsUniformlyCont_def
by simp
        have ∀t. t ∈ (ProdFunction(f,f)-V){x} <-> ⟨x,t⟩ ∈ ProdFunction(f,f)-(V)

          using image_def by auto
        with assms(1) x have ∀t. t: (ProdFunction(f,f)-V){x} ⟷ (t∈X
∧ ⟨fx,ft⟩ ∈ V)
          using prodFunVimage by auto
        with assms(1) as(1) have ∀t. t ∈ (ProdFunction(f,f)-V){x} ⟷
(t∈X ∧ ⟨xa,ft⟩: V)
          using apply_equality by auto
        with V(1) have ∀t. t ∈ (ProdFunction(f,f)-V){x} ⟷ (t∈X ∧ f(t)
∈ U) by auto
        with assms(1) U have ∀t. t ∈ (ProdFunction(f,f)-V){x} ⟷ t
∈ f-U
          using func1_1_L15 by simp
        hence f-U = (ProdFunction(f,f)-V){x} by blast
        with ent have f-(U) ∈ {V{x} . V ∈ Φ} by auto
        moreover
        have {⟨t,{V{t}.V∈Φ}⟩.t∈X} ∈ Pi(X,%t. {{V{t}.V∈Φ}}) unfolding
Pi_def function_def
          by auto
        ultimately have f-(U) ∈ {⟨t, {V  {t} . V ∈ Φ}⟩ . t ∈ X}(x) us-
ing x apply_equality
          by auto
      }
      ultimately show f-(U) ∈ UniformTopology(Φ,X) using uniftop_def_alt
by auto
    qed
  } then show thesis unfolding IsContinuous_def by simp
qed

end
```

94 Alternative definitions of uniformity

```
theory UniformSpace_ZF_2 imports UniformSpace_ZF
begin
```

The UniformSpace_ZF theory defines uniform spaces based on entourages (also called surroundings sometimes). In this theory we consider alternative definitions based of the notion of uniform covers and pseudometrics.

94.1 Uniform covers

Given a set X we can consider collections of subsets of X whose unions are equal to X. Any such collection is called a cover of X. We can define relation on the set of covers of X, called "star refinement" (definition below).

A collection of covers is a "family of uniform covers" if it is a filter with respect to the start refinement ordering. A member of such family is called a "uniform cover", but one has to remember that this notion has meaning only in the contexts a the whole family of uniform covers. Looking at a specific cover in isolation we can not say whether it is a uniform cover or not.

The set of all covers of X is called `Covers(X)`.

definition
  `Covers(X) ≡ {P ∈ Pow(Pow(X)). ⋃P = X}`

A cover of a nonempty set must have a nonempty member.

lemma `cover_nonempty:` **assumes** `X≠0 P ∈ Covers(X)`
  shows `∃U∈P. U≠0`
  using `assms` **unfolding** `Covers_def` **by** `blast`

A "star" of R with respect to $\mathcal{R}$ is the union of all $S \in \mathcal{R}$ that intersect R.

definition
  `Star(R,ℛ) ≡ ⋃{S∈ℛ. S∩R ≠ 0}`

An element of $\mathcal{R}$ is a subset of its star with respect to $\mathcal{R}$.

lemma `element_subset_star:` **assumes** `U∈P` **shows** `U ⊆ Star(U,P)`
  using `assms` **unfolding** `Star_def` **by** `auto`

An alternative formula for star of a singleton.

lemma `star_singleton:` **shows** `(⋃{V×V. V∈P}){x} = Star({x},P)`
  unfolding `Star_def` **by** `blast`

Star of a larger set is larger.

lemma `star_mono:` **assumes** `U⊆V` **shows** `Star(U,P) ⊆ Star(V,P)`
  using `assms` **unfolding** `Star_def` **by** `blast`

In particular, star of a set is larger than star of any singleton in that set.

corollary `star_single_mono:` **assumes** `x∈U` **shows** `Star({x},P) ⊆ Star(U,P)`
  using `assms star_mono` **by** `auto`

A cover $\mathcal{R}$ (of X) is said to be a "barycentric refinement" of a cover $\mathcal{C}$ iff for every $x \in X$ the star of $\{x\}$ in $\mathcal{R}$ is contained in some $C \in \mathcal{C}$.

definition
  `IsBarycentricRefinement (_ <`^{B}` _ 90)`
  where `P <`^{B}` Q ≡ ∀x∈⋃P.∃U∈Q.  Star({x},P) ⊆ U`

A cover is a barycentric refinement of the collection of stars of the singletons $\{x\}$ as x ranges over X.

lemma `singl_star_bary:`
  assumes `P ∈ Covers(X)` **shows** `P <`^{B}` {Star({x},P). x∈X}`

```
  using assms unfolding Covers_def IsBarycentricRefinement_def by blast
```

A cover $\mathcal{R}$ is a "star refinement" of a cover $\mathcal{C}$ iff for each $R \in \mathcal{R}$ there is a $C \in \mathcal{C}$ such that the star of R with respect to $\mathcal{R}$ is contained in C.

```
definition
  IsStarRefinement (_ <* _ 90)
  where P <* Q ≡ ∀U∈P.∃V∈Q. Star(U,P) ⊆ V
```

Every cover star-refines the trivial cover $\{X\}$.

```
lemma cover_stref_triv: assumes P ∈ Covers(X) shows P <* {X}
  using assms unfolding Star_def IsStarRefinement_def Covers_def by auto
```

Star refinement implies barycentric refinement.

```
lemma star_is_bary: assumes Q∈Covers(X) and Q <* P
  shows Q <^B P
proof -
  from assms(1) have ⋃Q = X unfolding Covers_def by simp
  { fix x assume x∈X
    with ‹⋃Q = X› obtain R where R∈Q and x∈R by auto
    with assms(2) obtain U where U∈P and Star(R,Q) ⊆ U
      unfolding IsStarRefinement_def by auto
    from ‹x∈R› ‹Star(R,Q) ⊆ U› have Star({x},Q) ⊆ U
      using star_single_mono by blast
    with ‹U∈P› have ∃U∈P. Star({x},Q) ⊆ U by auto
 } with ‹⋃Q = X› show thesis unfolding IsBarycentricRefinement_def
    by simp
qed
```

Barycentric refinement of a barycentric refinement is a star refinement.

```
lemma bary_bary_star:
  assumes P∈Covers(X) Q∈Covers(X) R∈Covers(X) P <^B Q Q <^B R X≠0
  shows P <* R
proof -
  { fix U assume U∈P
    { assume U = 0
      then have Star(U,P) = 0 unfolding Star_def by simp
      from assms(6,3) obtain V where V∈R using cover_nonempty by auto
      with ‹Star(U,P) = 0› have ∃V∈R. Star(U,P) ⊆ V by auto
    }
    moreover
    { assume U≠0
      then obtain x_0 where x_0∈U by auto
      with assms(1,2,5) ‹U∈P› obtain V where V∈R and Star({x_0},Q) ⊆
V
        unfolding Covers_def IsBarycentricRefinement_def by auto
      have Star(U,P) ⊆ V
      proof -
        { fix W assume W∈P and W∩U ≠ 0
```

```
        from ‹W∩U ≠ 0› obtain x where x∈W∩U by auto
        with assms(2) ‹U∈P› have  x∈⋃P by auto
        with assms(4) obtain C where C∈Q and Star({x},P) ⊆ C
          unfolding IsBarycentricRefinement_def by blast
        with ‹U∈P› ‹W∈P› ‹x∈W∩U› ‹x0∈U› ‹Star({x0},Q) ⊆ V› have W⊆V

          unfolding Star_def by blast
      } then show Star(U,P) ⊆ V unfolding Star_def by auto
    qed
    with ‹V∈R› have ∃V∈R. Star(U,P) ⊆ V by auto
  }
  ultimately have ∃V∈R. Star(U,P) ⊆ V by auto
} then show P <* R unfolding IsStarRefinement_def by simp
qed
```

The notion of a filter defined in `Topology_ZF_4` is not sufficiently general to use it to define uniform covers, so we write the conditions directly. A nonempty collection Θ of covers of X is a family of uniform covers if

a) if $\mathcal{R} \in \Theta$ and $\mathcal{C}$ is any cover of X such that $\mathcal{R}$ is a star refinement of $\mathcal{C}$, then $\mathcal{C} \in \Theta$.

b) For any $\mathcal{C}, \mathcal{D} \in \Theta$ there is some $\mathcal{R} \in \Theta$ such that $\mathcal{R}$ is a star refinement of both $\mathcal{C}$ and $\mathcal{R}$.

This departs slightly from the definition in Wikipedia that requires that Θ contains the trivial cover $\{X\}$. As we show in lemma `unicov_contains_trivial` below we don't loose anything by weakening the definition this way.

```
definition
  AreUniformCovers (_ {are uniform covers of} _ 90)
  where Θ {are uniform covers of} X ≡  Θ ⊆ Covers(X) ∧ Θ≠0 ∧
  (∀𝓡∈Θ.∀𝓒∈Covers(X). ((𝓡 <* 𝓒) ⟶ 𝓒∈Θ)) ∧
  (∀𝓒∈Θ.∀𝓓∈Θ.∃𝓡∈Θ.(𝓡 <* 𝓒) ∧ (𝓡 <* 𝓓))
```

A family of uniform covers contain the trivial cover $\{X\}$.

```
lemma unicov_contains_triv: assumes Θ {are uniform covers of} X
  shows {X} ∈ Θ
proof -
  from assms obtain 𝓡 where 𝓡∈Θ unfolding AreUniformCovers_def by blast
  with assms show thesis using cover_stref_triv
    unfolding AreUniformCovers_def Covers_def by auto
qed
```

If Θ are uniform covers of X then we can recover X from Θ by taking $\bigcup\bigcup\Theta$.

```
lemma space_from_unicov: assumes Θ {are uniform covers of} X shows
X = ⋃⋃Θ
proof
  from assms show X ⊆ ⋃⋃Θ using unicov_contains_triv
    unfolding AreUniformCovers_def by auto
```

```
  from assms show ⋃⋃Θ ⊆ X unfolding AreUniformCovers_def Covers_def
    by auto
qed
```

Every uniform cover has a star refinement.

```
lemma unicov_has_star_ref:
  assumes Θ {are uniform covers of} X and P∈Θ
  shows ∃Q∈Θ. (Q <* P)
  using assms unfolding AreUniformCovers_def by blast
```

In particular, every uniform cover has a barycentric refinement.

```
corollary unicov_has_bar_ref:
  assumes Θ {are uniform covers of} X and P∈Θ
  shows ∃Q∈Θ. (Q <^B P)
proof -
  from assms obtain Q where Q∈Θ and Q <* P
    using unicov_has_star_ref by blast
  with assms show thesis
    unfolding AreUniformCovers_def using star_is_bary by blast
qed
```

From the definition of uniform covers we know that if a uniform cover P is a star-refinement of a cover Q then Q is in a uniform cover. The next lemma shows that in order for Q to be a uniform cover it is sufficient that P is a barycentric refinement of Q.

```
lemma unicov_bary_cov:
  assumes Θ {are uniform covers of} X P∈Θ Q ∈ Covers(X) P <^B Q and
X≠0
  shows Q∈Θ
proof -
  from assms(1,2) obtain R where R∈Θ and R <^B P
    using unicov_has_bar_ref by blast
  from assms(1,2,3) ‹R∈Θ› have
    P ∈ Covers(X) Q ∈ Covers(X) R ∈ Covers(X)
    unfolding AreUniformCovers_def by auto
  with assms(1,3,4,5) ‹R∈Θ› ‹R <^B P› show thesis
    using bary_bary_star unfolding AreUniformCovers_def by auto
qed
```

A technical lemma to simplify proof of the `uniformity_from_unicov` theorem.

```
lemma star_ref_mem: assumes U∈P P<*Q and ⋃{W×W. W∈Q} ⊆ A
  shows U×U ⊆ A
proof -
  from assms(1,2) obtain W where W∈Q and ⋃{S∈P. S∩U ≠ 0} ⊆ W
    unfolding IsStarRefinement_def Star_def by auto
  with assms(1,3) show U×U ⊆ A by blast
qed
```

An identity related to square (in the sense of composition) of a relation of the form $\bigcup\{U \times U : U \in P\}$. I am amazed that Isabelle can see that this is true without an explicit proof, I can't.

```
lemma rel_square_starr: shows
  (⋃{U×U. U∈P}) O (⋃{U×U. U∈P}) = ⋃{U×Star(U,P). U∈P}
  unfolding Star_def by blast
```

An identity similar to `rel_square_starr` but with `Star` on the left side of the Cartesian product:

```
lemma rel_square_starl: shows
  (⋃{U×U. U∈P}) O (⋃{U×U. U∈P}) = ⋃{Star(U,P)×U. U∈P}
  unfolding Star_def by blast
```

A somewhat technical identity about the square of a symmetric relation:

```
lemma rel_sq_image:
  assumes W = converse(W)  domain(W) ⊆ X
  shows Star({x},{W{t}. t∈X}) = (W O W){x}
proof
  have I: Star({x},{W{t}. t∈X}) = ⋃{S∈{W{t}. t∈X}. x∈S}
    unfolding Star_def by auto
  { fix y assume y ∈ Star({x},{W{t}. t∈X})
    with I obtain S where y∈S x∈S S ∈ {W{t}. t∈X} by auto
    from ‹S ∈ {W{t}. t∈X}› obtain t where t∈X and S = W{t}
      by auto
    with ‹x∈S› ‹y∈S› have ⟨t,x⟩ ∈ W and ⟨t,y⟩ ∈ W
      by auto
    from ‹⟨t,x⟩ ∈ W› have ⟨x,t⟩ ∈ converse(W) by auto
    with assms(1) ‹⟨t,y⟩ ∈ W› have y ∈ (W O W){x}
      using rel_compdef by auto
  } then show Star({x},{W{t}. t∈X}) ⊆ (W O W){x}
    by blast
  { fix y assume y∈(W O W){x}
    then obtain t where ⟨x,t⟩ ∈ W and ⟨t,y⟩ ∈ W
      using rel_compdef by auto
    from assms(2) ‹⟨t,y⟩ ∈ W› have t∈X by auto
    from ‹⟨x,t⟩ ∈ W› have ⟨t,x⟩ ∈ converse(W) by auto
    with assms(1) I ‹⟨t,y⟩ ∈ W› ‹t∈X› have y ∈ Star({x},{W{t}. t∈X})
      by auto
  } then show (W O W){x} ⊆ Star({x},{W{t}. t∈X})
    by blast
qed
```

Given a family of uniform covers of X we can create a uniformity on X by taking the supersets of $\bigcup\{A \times A : A \in P\}$ as P ranges over the uniform covers. The next definition specifies the operation creating entourages from uniform covers.

definition

```
  UniformityFromUniCov(X,Θ) ≡ Supersets(X×X,{⋃{U×U. U∈P}. P∈Θ})
```

For any member P of a cover Θ the set $\bigcup\{U \times U : U \in P\}$ is a member of `UniformityFromUniCov(X,Θ)`.

```
lemma basic_unif: assumes Θ ⊆ Covers(X) P∈Θ
  shows ⋃{U×U. U∈P} ∈ UniformityFromUniCov(X,Θ)
  using assms unfolding UniformityFromUniCov_def Supersets_def Covers_def
  by blast
```

If Θ is a family of uniform covers of X then `UniformityFromUniCov(X,Θ)` is a uniformity on X

```
theorem uniformity_from_unicov:
  assumes Θ {are uniform covers of} X X≠0
  shows UniformityFromUniCov(X,Θ) {is a uniformity on} X
proof -
  let Φ = UniformityFromUniCov(X,Θ)
  have Φ {is a filter on} (X×X)
  proof -
    have 0 ∉ Φ
    proof -
      { assume 0 ∈ Φ
        then obtain P where P∈Θ and 0 = ⋃{U×U. U∈P}
          unfolding UniformityFromUniCov_def Supersets_def by auto
        hence ⋃P = 0 by auto
        with assms ‹P∈Θ› have False unfolding AreUniformCovers_def Covers_def
          by auto
      } thus thesis by auto
    qed
    moreover have X×X ∈ Φ
    proof -
      from assms have X×X ∈ {⋃{U×U. U∈P}. P∈Θ}
        using unicov_contains_triv unfolding AreUniformCovers_def
        by auto
      then show thesis unfolding Supersets_def UniformityFromUniCov_def

        by blast
    qed
    moreover have Φ ⊆ Pow(X×X)
      unfolding UniformityFromUniCov_def Supersets_def by auto
    moreover have ∀A∈Φ.∀B∈Φ. A∩B ∈ Φ
    proof -
      { fix A B assume A∈Φ B∈Φ
        then have A∩B ⊆ X×X unfolding UniformityFromUniCov_def Supersets_def
          by auto
        from ‹A∈Φ› ‹B∈Φ› obtain P_A P_B where
          P_A∈Θ P_B∈Θ and I:⋃{U×U. U∈P_A} ⊆ A  ⋃{U×U. U∈P_B} ⊆ B
          unfolding UniformityFromUniCov_def Supersets_def by auto
        from assms(1) ‹P_A∈Θ› ‹P_B∈Θ› obtain P
          where P∈Θ and P<*P_A and P<*P_B
```

```
        unfolding AreUniformCovers_def by blast
      have ⋃{U×U. U∈P} ⊆ A∩B
      proof -
        { fix U assume U∈P
          with ‹P<*P_A› ‹P<*P_B› I have U×U ⊆ A and U×U ⊆ B
            using star_ref_mem by auto
        } thus thesis by blast
      qed
      with ‹A∩B ⊆ X×X› ‹P∈Θ› have A∩B ∈ Φ
        unfolding Supersets_def UniformityFromUniCov_def by auto
    } thus thesis by auto
  qed
  moreover have
    ∀B∈Φ.∀C∈Pow(X×X). B⊆C ⟶ C∈Φ
  proof -
    { fix B C assume B∈Φ C∈Pow(X×X) B⊆C
      from ‹B∈Φ› obtain P_B where  ⋃{U×U. U∈P_B} ⊆ B P_B∈Θ
        unfolding UniformityFromUniCov_def Supersets_def by auto
      with ‹C∈Pow(X×X)› ‹B⊆C› have C∈Φ
        unfolding UniformityFromUniCov_def Supersets_def by blast
    } thus thesis by auto
  qed
  ultimately show thesis unfolding IsFilter_def by simp
qed
moreover have ∀A∈Φ. id(X) ⊆ A ∧ (∃B∈Φ. B O B ⊆ A) ∧ converse(A)
∈ Φ
proof
  fix A assume A∈Φ
  then obtain P where ⋃{U×U. U∈P} ⊆ A P∈Θ
    unfolding UniformityFromUniCov_def Supersets_def by auto
  have id(X)⊆A
  proof -
    from assms(1) ‹P∈Θ› have ⋃P = X unfolding AreUniformCovers_def
Covers_def
      by auto
    with ‹⋃{U×U. U∈P} ⊆ A› show thesis by auto
  qed
  moreover have ∃B∈Φ. B O B ⊆ A
  proof -
    from assms(1) ‹P∈Θ› have ⋃{U×U. U∈P} ∈ Φ
      unfolding AreUniformCovers_def Covers_def UniformityFromUniCov_def
Supersets_def
      by auto
    from assms(1) ‹P∈Θ› obtain Q where Q∈Θ and  Q <* P using unicov_has_star_ref
      by blast
    let B = ⋃{U×U. U∈Q}
    from assms(1) ‹Q∈Θ› have B ∈ Φ
      unfolding AreUniformCovers_def Covers_def UniformityFromUniCov_def
Supersets_def
```

```
      by auto
    moreover have B O B ⊆ A
    proof -
      have II: B O B = ⋃{U×Star(U,Q). U∈Q} using rel_square_starr

        by simp
      have ∀U∈Q. ∃V∈P. U×Star(U,Q) ⊆ V×V
      proof
        fix U assume U∈Q
        with ‹Q <* P› obtain V where V∈P and Star(U,Q) ⊆ V
          unfolding IsStarRefinement_def by blast
        with ‹U∈Q› have V∈P and U×Star(U,Q) ⊆ V×V using element_subset_star
          by auto
        thus ∃V∈P. U×Star(U,Q) ⊆ V×V by auto
      qed
      hence ⋃{U×Star(U,Q). U∈Q} ⊆ ⋃{V×V. V∈P} by blast
      with ‹⋃{V×V. V∈P} ⊆ A› have ⋃{U×Star(U,Q). U∈Q} ⊆ A by blast
      with II show thesis by simp
    qed
    ultimately show thesis by auto
  qed
  moreover from ‹A∈Φ› ‹P∈Θ› ‹⋃{U×U. U∈P} ⊆ A› have converse(A)
∈ Φ
    unfolding AreUniformCovers_def UniformityFromUniCov_def Supersets_def

    by auto
  ultimately show id(X) ⊆ A ∧ (∃B∈Φ. B O B ⊆ A) ∧ converse(A) ∈ Φ

    by simp
 qed
 ultimately show Φ  {is a uniformity on} X unfolding IsUniformity_def

  by simp
qed
```

Given a uniformity Φ on X we can create a family of uniform covers by taking the collection of covers P for which there exist an entourage $U \in \Phi$ such that for each $x \in X$, there is an $A \in P$ such that $U(\{x\}) \subseteq A$. The next definition specifies the operation of creating a family of uniform covers from a uniformity.

```
definition
 UniCovFromUniformity(X,Φ) ≡ {P∈Covers(X). ∃U∈Φ.∀x∈X.∃A∈P. U({x})
⊆ A}
```

When we convert the quantifiers into unions and intersections in the definition of `UniCovFromUniformity` we get an alternative definition of the operation that creates a family of uniform covers from a uniformity. Just a curiosity, not used anywhere.

```
lemma UniCovFromUniformityDef: assumes X≠0
  shows UniCovFromUniformity(X,Φ) = (⋃U∈Φ.⋂x∈X. {P∈Covers(X). ∃A∈P.
U({x}) ⊆ A})
proof -
  have {P∈Covers(X). ∃U∈Φ.∀x∈X.∃A∈P. U({x}) ⊆ A} =
  (⋃U∈Φ.⋂x∈X. {P∈Covers(X). ∃A∈P. U({x}) ⊆ A})
  proof
  { fix P assume P∈{P∈Covers(X). ∃U∈Φ.∀x∈X.∃A∈P. U({x}) ⊆ A}
    then have P∈Covers(X) and ∃U∈Φ.∀x∈X.∃A∈P. U({x}) ⊆ A by auto
    then obtain U where U∈Φ and ∀x∈X.∃A∈P. U({x}) ⊆ A by auto
    with assms ‹P∈Covers(X)› have P ∈ (⋂x∈X. {P∈Covers(X). ∃A∈P. U({x})
⊆ A})
      by auto
    with ‹U∈Φ›  have P∈(⋃U∈Φ.⋂x∈X. {P∈Covers(X). ∃A∈P. U({x}) ⊆
A})
      by blast
  } then show
    {P∈Covers(X). ∃U∈Φ.∀x∈X.∃A∈P. U({x}) ⊆ A} ⊆
    (⋃U∈Φ.⋂x∈X. {P∈Covers(X). ∃A∈P. U({x}) ⊆ A})
    using subset_iff by simp
  { fix P assume P∈(⋃U∈Φ.⋂x∈X. {P∈Covers(X). ∃A∈P. U({x}) ⊆ A})
    then obtain U where U∈Φ P ∈ (⋂x∈X. {P∈Covers(X). ∃A∈P. U({x})
⊆ A})
      by auto
    with assms have P∈Covers(X) and ∀x∈X.∃A∈P. U({x}) ⊆ A by auto
    with ‹U∈Φ› have P∈{P∈Covers(X). ∃U∈Φ.∀x∈X.∃A∈P. U({x}) ⊆ A}
      by auto
  } then show (⋃U∈Φ.⋂x∈X. {P∈Covers(X). ∃A∈P. U({x}) ⊆ A}) ⊆
    {P∈Covers(X). ∃U∈Φ.∀x∈X.∃A∈P. U({x}) ⊆ A} by auto
  qed
  then show thesis unfolding UniCovFromUniformity_def by simp
qed
```

If Φ is a (diagonal) uniformity on X, then covers of the form $\{W\{x\} : x \in X\}$ are members of `UniCovFromUniformity(X,Φ)`.

```
lemma cover_image:
  assumes Φ {is a uniformity on} X W∈Φ
  shows {W{x}. x∈X} ∈ UniCovFromUniformity(X,Φ)
proof -
  let P = {W{x}. x∈X}
  have P ∈ Covers(X)
  proof -
    from assms have W ⊆ X×X and P ∈ Pow(Pow(X))
      using entourage_props(1) by auto
    moreover have ⋃P = X
    proof
      from ‹W ⊆ X×X› show ⋃P ⊆ X by auto
      from assms show X ⊆ ⋃P using neigh_not_empty(2) by auto
    qed
```

```
    ultimately show thesis unfolding Covers_def by simp
  qed
  moreover from assms(2) have ∃W∈Φ. ∀x∈X. ∃A∈P. W{x} ⊆ A
    by auto
  ultimately show thesis unfolding UniCovFromUniformity_def
    by simp
qed
```

If Φ is a (diagonal) uniformity on X, then every two elements of UniCovFromUniformity(X,Φ) have a common barycentric refinement.

```
lemma common_bar_refinemnt:
  assumes
    Φ {is a uniformity on} X
    Θ = UniCovFromUniformity(X,Φ)
    𝒞∈Θ 𝒟∈Θ
  shows ∃ℛ∈Θ.(ℛ <^B 𝒞) ∧ (ℛ <^B 𝒟)
proof -
  from assms(2,3) obtain U where U∈Φ and I: ∀x∈X.∃C∈𝒞. U{x} ⊆ C
    unfolding UniCovFromUniformity_def by auto
  from assms(2,4) obtain V where V∈Φ and II: ∀x∈X.∃D∈𝒟. V{x} ⊆ D
    unfolding UniCovFromUniformity_def by auto
  from assms(1) ‹U∈Φ› ‹V∈Φ› have U∩V ∈ Φ
    unfolding IsUniformity_def IsFilter_def by auto
  with assms(1) obtain W where W∈Φ and W O W ⊆ U∩V and W=converse(W)
    using half_size_symm by blast
  from assms(1) ‹W∈Φ› have domain(W) ⊆ X
    unfolding IsUniformity_def IsFilter_def by auto
  let P = {W{t}. t∈X}
  have P∈Θ P <^B 𝒞 P <^B 𝒟
  proof -
    from assms(1,2) ‹W∈Φ› show P∈Θ using cover_image by simp
    with assms(2) have ⋃P = X unfolding UniCovFromUniformity_def Covers_def
      by simp
    { fix x assume x∈X
      from ‹W=converse(W)› ‹domain(W) ⊆ X› ‹W O W ⊆ U∩V›
      have Star({x},P) ⊆ U{x} and Star({x},P) ⊆ V{x}
        using rel_sq_image by auto
      from ‹x∈X› I obtain C where C∈𝒞 and U{x} ⊆ C
        by auto
      with ‹Star({x},P) ⊆ U{x}› ‹C∈𝒞› have ∃C∈𝒞. Star({x},P) ⊆ C
        by auto
      moreover
      from ‹x∈X› II obtain D where D∈𝒟 and V{x} ⊆ D
        by auto
      with ‹Star({x},P) ⊆ V{x}› ‹D∈𝒟› have ∃D∈𝒟. Star({x},P) ⊆ D
        by auto
      ultimately have ∃C∈𝒞. Star({x},P) ⊆ C and ∃D∈𝒟. Star({x},P) ⊆
D
        by auto
```

```
    } hence ∀x∈X. ∃C∈𝒞. Star({x},P) ⊆ C and ∀x∈X.∃D∈𝒟. Star({x},P)
⊆ D
      by auto
    with ‹⋃P = X› show P <^B 𝒞 and P <^B 𝒟
      unfolding IsBarycentricRefinement_def by auto
  qed
  thus thesis by auto
qed
```

If Φ is a (diagonal) uniformity on X, then every element of UniCovFromUniformity(X,Φ) has a barycentric refinement there.

```
corollary bar_refinement_ex:
  assumes Φ {is a uniformity on} X Θ = UniCovFromUniformity(X,Φ) 𝒞 ∈
Θ
  shows ∃ℛ∈Θ. (ℛ <^B 𝒞)
  using assms common_bar_refinemnt by blast
```

If Φ is a (diagonal) uniformity on X, then UniCovFromUniformity(X,Φ) is a family of uniform covers.

```
theorem unicov_from_uniformity: assumes Φ {is a uniformity on} X and
X≠0
  shows UniCovFromUniformity(X,Φ) {are uniform covers of} X
proof -
  let Θ = UniCovFromUniformity(X,Φ)
  from assms(1) have Θ ⊆ Covers(X) unfolding UniCovFromUniformity_def

    by auto
  moreover
  from assms(1) have {X} ∈ Θ
    unfolding Covers_def IsUniformity_def IsFilter_def UniCovFromUniformity_def
    by auto
  hence Θ ≠ 0 by auto
  moreover have ∀ℛ∈Θ.∀𝒞∈Covers(X). ((ℛ <* 𝒞) ⟶ 𝒞∈Θ)
  proof -
    { fix ℛ 𝒞 assume ℛ∈Θ 𝒞∈Covers(X) ℛ <* 𝒞
      have 𝒞∈Θ
      proof -
        from ‹ℛ∈Θ› obtain U where U∈Φ and I: ∀x∈X.∃R∈ℛ. U({x}) ⊆
R
          unfolding UniCovFromUniformity_def by auto
        { fix x assume x∈X
          with I obtain R where R∈ℛ and U({x}) ⊆ R by auto
          from ‹R∈ℛ› ‹ℛ <* 𝒞› obtain C where C∈𝒞 and Star(R,ℛ) ⊆
C
            unfolding IsStarRefinement_def by auto
          with ‹U({x}) ⊆ R› ‹R∈ℛ› have U({x}) ⊆ C
            using element_subset_star by blast
          with ‹C∈𝒞› have ∃C∈𝒞. U({x}) ⊆ C by auto
        } hence ∀x∈X.∃C∈𝒞. U({x}) ⊆ C by auto
```

```
      with ‹U∈Φ› ‹𝒞∈Covers(X)› show thesis unfolding UniCovFromUniformity_def
        by auto
    qed
  } thus thesis by auto
qed
moreover have ∀𝒞∈Θ.∀𝒟∈Θ.∃ℛ∈Θ.(ℛ <* 𝒞) ∧ (ℛ <* 𝒟)
proof -
  { fix 𝒞 𝒟 assume 𝒞∈Θ 𝒟∈Θ
    with assms(1) obtain P where P∈Θ and P <^B 𝒞 P <^B 𝒟
      using common_bar_refinemnt by blast
    from assms(1) ‹P∈Θ› obtain ℛ where ℛ∈Θ and ℛ <^B P
      using bar_refinement_ex by blast
    from ‹ℛ∈Θ› ‹P∈Θ› ‹𝒞∈Θ› ‹𝒟∈Θ› have
      P ∈ Covers(X) ℛ ∈ Covers(X) 𝒞 ∈ Covers(X) 𝒟 ∈ Covers(X)
      unfolding UniCovFromUniformity_def by auto
    with assms(2) ‹ℛ <^B P› ‹P <^B 𝒞› ‹P <^B 𝒟› have ℛ <* 𝒞 and ℛ <*
𝒟
      using bary_bary_star by auto
    with ‹ℛ∈Θ› have ∃ℛ∈Θ.(ℛ <* 𝒞) ∧ (ℛ <* 𝒟) by auto
  } thus thesis by simp
qed
ultimately show thesis unfolding AreUniformCovers_def by simp
qed
```

The UniCovFromUniformity operation is the inverse of UniformityFromUniCov.

```
theorem unicov_from_unif_inv: assumes Θ {are uniform covers of} X X≠0
  shows UniCovFromUniformity(X,UniformityFromUniCov(X,Θ)) = Θ
proof
  let Φ = UniformityFromUniCov(X,Θ)
  let L = UniCovFromUniformity(X,Φ)
  from assms have I: Φ {is a uniformity on} X
    using uniformity_from_unicov by simp
  with assms(2) have II: L {are uniform covers of} X
    using unicov_from_uniformity by simp
  { fix P assume P∈L
    with I obtain Q where Q∈L and Q <^B P
      using bar_refinement_ex by blast
    from ‹Q∈L› obtain U where U∈Φ and III:∀x∈X.∃A∈Q. U{x} ⊆ A
      unfolding UniCovFromUniformity_def by auto
    from ‹U∈Φ› have U ∈ Supersets(X×X,{⋃{U×U. U∈P}. P∈Θ})
      unfolding UniformityFromUniCov_def by simp
    then obtain B where B⊆X×X B⊆U and ∃C∈{⋃{U×U. U∈P}. P∈Θ}. C⊆B

      unfolding Supersets_def by auto
    then obtain C where C ∈ {⋃{U×U. U∈P}. P∈Θ} and C⊆B by auto
    then obtain R where R∈Θ and C = ⋃{V×V. V∈R} by auto
    with ‹C⊆B› ‹B⊆U› have ⋃{V×V. V∈R} ⊆ U by auto
    from assms(1) II ‹P∈L› ‹Q∈L› ‹R∈Θ› have
      IV: P∈Covers(X) Q∈Covers(X) R∈Covers(X)
```

```
      unfolding AreUniformCovers_def by auto
    have R <^B Q
    proof -
      { fix x assume x∈X
        with III obtain A where A∈Q and U{x} ⊆ A by auto
        with ‹⋃{V×V. V∈R} ⊆ U› have (⋃{V×V. V∈R}){x} ⊆ A
          by auto
        with ‹A∈Q› have ∃A∈Q. Star({x},R) ⊆ A using star_singleton by
auto
      } then have ∀x∈X. ∃A∈Q. Star({x},R) ⊆ A by simp
      moreover from ‹R∈Covers(X)› have ⋃R = X unfolding Covers_def
        by simp
      ultimately show thesis unfolding IsBarycentricRefinement_def
        by simp
    qed
    with assms(2) ‹Q <^B P› IV have R <* P using bary_bary_star by simp
    with assms(1) ‹R∈Θ› ‹P∈Covers(X)› have P∈Θ
      unfolding AreUniformCovers_def by simp
  } thus L⊆Θ by auto
  { fix P assume P∈Θ
    with assms(1) have P ∈ Covers(X)
      unfolding AreUniformCovers_def by auto
    from assms(1) ‹P∈Θ› obtain Q where Q ∈ Θ and Q <^B P
      using unicov_has_bar_ref by blast
    let A = ⋃{V×V. V∈Q}
    have A ∈ Φ
    proof -
      from assms(1) ‹Q∈Θ› have A ⊆ X×X and A ∈ {⋃{V×V. V∈Q}. Q∈Θ}
        unfolding AreUniformCovers_def Covers_def by auto
      then show thesis
        using superset_gen unfolding UniformityFromUniCov_def
        by auto
    qed
    with I obtain B where B∈Φ B O B ⊆ A and B=converse(B)
      using half_size_symm by blast
    let R = {B{x}. x∈X}
    from I II ‹B∈Φ› have R∈L and ⋃R =X
      using cover_image unfolding UniCovFromUniformity_def Covers_def
      by auto
    have R <^B P
    proof -
      { fix x assume x∈X
        from assms(1) ‹Q ∈ Θ› have ⋃Q = X
          unfolding AreUniformCovers_def Covers_def by auto
        with ‹Q <^B P› ‹x∈X› obtain C where C∈P and Star({x},Q) ⊆ C
          unfolding IsBarycentricRefinement_def by auto
        from ‹B=converse(B)› I ‹B∈Φ› have Star({x},R) = (B O B){x}
          using  uni_domain rel_sq_image by auto
        moreover from  ‹(B O B) ⊆ A› have (B O B){x} ⊆ A{x} by blast
```

```
        moreover have A{x} = Star({x},Q) using star_singleton by simp
        ultimately have Star({x},R) ⊆ Star({x},Q) by auto
        with ‹Star({x},Q) ⊆ C› ‹C∈P› have ∃C∈P. Star({x},R) ⊆ C
          by auto
      } with ‹⋃R = X› show thesis unfolding IsBarycentricRefinement_def
        by auto
    qed
    with assms(2) II ‹P ∈ Covers(X)› ‹R∈L› ‹R <^B P› have P∈L
      using unicov_bary_cov by simp
  } thus Θ⊆L by auto
qed
```

The UniformityFromUniCov operation is the inverse of UniCovFromUniformity.

```
theorem unif_from_unicov_inv: assumes Φ {is a uniformity on} X X≠0
  shows UniformityFromUniCov(X,UniCovFromUniformity(X,Φ)) = Φ
proof
  let Θ = UniCovFromUniformity(X,Φ)
  let L = UniformityFromUniCov(X,Θ)
  from assms have I: Θ {are uniform covers of} X
    using unicov_from_uniformity by simp
  with assms have II: L {is a uniformity on} X
    using uniformity_from_unicov by simp
  { fix A assume A∈Φ
    with assms(1) obtain B where B∈Φ B O B ⊆ A and B = converse(B)
      using half_size_symm by blast
    from assms(1) ‹A∈Φ› have A ⊆ X×X using uni_domain(1)
      by simp
    let P = {B{x}. x∈X}
    from assms(1) ‹B∈Φ› have P∈Θ using cover_image
      by simp
    let C = ⋃{U×U. U∈P}
    from I ‹P∈Θ› have C∈L
      unfolding AreUniformCovers_def using basic_unif by blast
    from assms(1) ‹B∈Φ› ‹B = converse(B)› ‹B O B ⊆ A› have C ⊆ A
      using uni_domain(2) symm_sq_prod_image by simp
    with II ‹A ⊆ X×X› ‹C∈L› have A∈L
      unfolding IsUniformity_def IsFilter_def by simp
  } thus Φ⊆L by auto
  { fix A assume A∈L
    with II have A ⊆ X×X using entourage_props(1) by simp
    from ‹A∈L› obtain P where P∈Θ and ⋃{U×U. U∈P} ⊆ A
      unfolding UniformityFromUniCov_def Supersets_def by blast
    from ‹P∈Θ› obtain B where B∈Φ and III: ∀x∈X. ∃V∈P. B{x} ⊆ V
      unfolding UniCovFromUniformity_def by auto
    have B⊆A
    proof -
      from assms(1) ‹B∈Φ› have B ⊆ ⋃{B{x}×B{x}. x∈X}
        using entourage_props(1,2) refl_union_singl_image by simp
      moreover have ⋃{B{x}×B{x}. x∈X} ⊆ A
```

```
      proof -
        { fix x assume x∈X
          with III obtain V where V∈P and B{x} ⊆ V by auto
          hence B{x}×B{x} ⊆ ⋃{U×U. U∈P} by auto
        } hence ⋃{B{x}×B{x}. x∈X} ⊆ ⋃{U×U. U∈P} by blast
        with ‹⋃{U×U. U∈P} ⊆ A› show thesis by blast
      qed
      ultimately show thesis by auto
    qed
    with assms(1) ‹B∈Φ› ‹A ⊆ X×X› have A∈Φ
      unfolding IsUniformity_def IsFilter_def by simp
  } thus L⊆Φ by auto
qed

end
```

95 Real valued metric spaces

```
theory MetricSpace_ZF_1 imports Real_ZF_2
begin
```

The development of metric spaces in IsarMathLib is different from the usual treatment of the subject because the notion of a metric (or a pseudometric) is defined in the `MetricSpace_ZF` theory a more generally as a function valued in an ordered loop. This theory file brings the subject closer to the standard way by specializing that general definition to the usual special case where the value of the metric are nonnegative real numbers.

95.1 Real valued metric scapes: context and notation

The `reals` context (locale) defined in the `Real_ZF_2` theory fixes a model of reals (i.e. a complete ordered field) and defines notation for things like zero, one, the set of positive numbers, absolute value etc. For metric spaces we reuse the notation defined there.

The `rpmetric_space` locale extends the `reals` locale, adding the carrier X of the metric space and the metric $\lceil$ to the context, together with the assumption that $\lceil : X \times X \to \mathbb{R}^+$ is a pseudo metric. We choose to denote the disk in X with center c and radius r as `ball(c,r)` As in the `pmetric_space` locale we define the τ to be the metric topology, i.e. the topology induced by the (real valued) pseudometric $\lceil$. An alternative would be to define the `rpmetric_space` as an extension of the `rpmetric_space` context, but that is in turn an extension of the `loop1` locale that defines notation for left and right division which which do not want in the context of real numbers.

```
locale rpmetric_space = reals +
  fixes X and d
```

```
  assumes pmetricAssum: IsApseudoMetric(d,X,ℝ,Add,ROrd)
  fixes ball
  defines ball_def [simp]: ball(c,r) ≡ Disk(X,d,ROrd,c,r)
  fixes pmettop (τ)
  defines pmettop [simp]: τ ≡ MetricTopology(X,ℝ,Add,ROrd,d)
  fixes interior (int)
  defines interior_def [simp]: int(D) ≡ Interior(D,τ)
  fixes cl
  defines cl_def [simp]: cl(D) ≡ Closure(D,τ)
```

The propositions proven in the `pmetric_space` context defined in `Metric_Space_ZF` theory are valid in the `rpmetric_space` context.

```
lemma (in rpmetric_space) pmetric_space_rpmetric_space_valid:
  shows pmetric_space(ℝ,Add,ROrd,d,X)
  unfolding pmetric_space_def pmetric_space_axioms_def loop1_def
  using pmetricAssum reals_loop by simp
```

The context `rpmetric_space` is a special case of context `pmetric_space` where the fixed objects in `pmetric_space` map to (in the order defined in `pmetric_space`) the set of real numbers, real addition, the order relation on reals, the strict order relation on reals, the set of non-negative reals and the set of positive reals. The metrics d maps to the real metrics d, the carrier of the metric space X is still X, and the `disks` from `pmetric_space` are now called `balls` in `rpmetric_space`. The notation for right and left division from `rpmetric_space` is not used in `pmetric_space`.

```
sublocale rpmetric_space < pmetric_space
  ℝ Add ROrd 0 realadd lesseq sless nonnegative positiveset
  λx y. LeftDiv(ℝ,Add)⟨x,y⟩
  λx y. RightDiv(ℝ,Add)⟨y,x⟩
  d X ball
  using pmetric_space_rpmetric_space_valid by simp_all
```

The `rmetric_space` locale (context) specializes the the `rpmetric_space` context by adding the assumption of identity of indiscernibles.

```
locale rmetric_space = rpmetric_space +
  assumes ident_indisc: ∀x∈X. ∀y∈Y. d⟨x,y⟩ = 0 ⟶ x=y
```

The propositions proven in the `metric_space` context defined in `Metric_Space_ZF` theory are valid in the `rmetric_space` context.

```
lemma (in rmetric_space) metric_space_rmetric_space_valid:
  shows metric_space(ℝ,Add,ROrd,d,X)
  unfolding metric_space_def metric_space_axioms_def
  using pmetric_space_rpmetric_space_valid ident_indisc
  by simp
```

The `rmetric_space` context is a special case of the `metric_space` context, with fixed objects mapping the same as in the mapping between `rpmetric_space` and `pmetric_space` above.

```
sublocale rmetric_space < metric_space
   ℝ Add ROrd 0 realadd lesseq sless nonnegative positiveset
  λx y. LeftDiv(ℝ,Add)⟨x,y⟩
  λx y. RightDiv(ℝ,Add)⟨y,x⟩
  d X ball
proof
  from ident_indisc show ∀x∈X. ∀y∈X. d  ⟨x, y⟩ = TheNeutralElement(ℝ,
Add) ⟶ x = y
    by simp
qed
```

95.2 Real valued metric spaces are Hausdorff as topological spaces

The usual (real-valued) metric spaces are a special case of ordered loop valued metric spaces defined in the `MetricSpace_ZF` theory, hence they are T_2 as topological spaces. Below we repeat the major theorems of `MetricSpace_ZF` theory specialized the standard setting of real valued metrics.

Since in the `rpmetric_space` context 𝔡 is a pseudometrics the (real valued) metric topology indeed a topology.

```
theorem (in rpmetric_space) rpmetric_is_top:
  shows  τ {is a topology}
  using rord_down_directs pmetric_is_top by simp
```

The collection of open disks (caled `balls` in the `rpmetric_space` context is a base for the (real valued) metric topology.

```
theorem (in rpmetric_space) rdisks_are_base:
  shows (⋃c∈X. {ball(c,R). R∈ℝ₊}) {is a base for} τ
  using rord_down_directs disks_are_base by simp
```

X is the carrier of the (real valued) metric topology.

```
theorem (in rpmetric_space) rmetric_top_carrier: shows  ⋃τ = X
  using rord_down_directs metric_top_carrier by simp
```

The topology generated by a (real valued) metric is Hausdorff (i.e. T_2).

```
theorem (in rmetric_space) rmetric_space_T2: shows τ {is T₂}
  using rord_down_directs metric_space_T2 by simp
```

95.3 Real valued (pseudo)metric spaces as uniform spaces

The ordered loop valued pseudometric spaces are uniform spaces. In this section we specialize major propositions from that context to the real valued pseudometric.

In the `MetricSpace_ZF` theory we define a property `IsHalfable` of an ordered loop that states that for every positive element b_1 of the loop there is another (positive) one b_2 such that $b_2 + b_2 \leq b_1$. This property is needed for the ordered loop valued pseudometric space to be a uniform space. In the next lemma we show that real numbers satisfy this property.

```
lemma (in reals) pos_reals_halfable: shows IsHalfable(ℝ,Add,ROrd)
proof -
  { fix x assume x∈ℝ+
    let y = (2⁻¹)·x
    from ‹x∈ℝ+› have x∈ℝ and y∈ℝ+
      using element_pos pos_mul_closed ord_ring_less_members one_half_pos(2)
      by simp_all
    from ‹x∈ℝ› have (2⁻¹+2⁻¹)·x = x using half_half_one(2) Ring_ZF_1_L3(6)
by simp
    with ‹x∈ℝ› have y+y ≤ x
      using ord_ring_less_members ring_oper_distr(2) one_half_pos(2) ring_ord_refl
      by auto
    with ‹y∈ℝ+› have ∃y∈ℝ+. y + y ≤ x by auto
  } then show thesis unfolding IsHalfable_def by simp
qed
```

In the `rpmetric_space` we will write `UniformGauge(X,ℝ,Add,ROrd,d)` i.e. $\{\lceil^{-1}([0,b] : b \in \mathbb{R}_+\}$ as $\mathfrak{U}$.

```
abbreviation (in rpmetric_space) rgauge (𝔘) where 𝔘 ≡ UniformGauge(X,ℝ,Add,ROrd,d)
```

$\mathfrak{U}$ is a fundamental system of entourages, hence its supersets form a uniformity on X and hence those supersets define a topology on X. This is a special case of the theorem `metric_gauge_base` from the `Metric_Space_ZF` theory but instead an ordered loop we have real numbers, so all the premises are automatically satisfied, except for the one of X being nonempty.

```
theorem (in rpmetric_space) metric_gauge_base: assumes X≠∅
  shows
    𝔘 {is a uniform base on} X
    Supersets(X×X,𝔘) {is a uniformity on} X
    UniformTopology(Supersets(X×X,𝔘),X) {is a topology}
    ⋃UniformTopology(Supersets(X×X,𝔘),X) = X
  using assms rord_down_directs pos_non_empty pos_reals_halfable metric_gauge_base
  by simp_all
```

The topology generated by open disks is the same as the one coming from the the unifomity consisting of supersets of sets in $\mathfrak{U}$.

```
theorem (in rpmetric_space) rmetric_top_is_uniform_top: assumes X≠∅
  shows τ = UniformTopology(Supersets(X×X,𝔘),X)
  using assms rord_down_directs pos_non_empty pos_reals_halfable metric_top_is_uniform_top
  by simp

end
```

96 Uniformity defined by a a collection of pseudometrics

theory `MetricUniform_ZF` **imports** `FinOrd_ZF_1 MetricSpace_ZF`

begin

Note: this theory is a work in progress. The approach taken is probably not the right one. The right approach is through the notion of least upper bound of a collection of uniformities.

In the `MetricSpace_ZF` we show how a single (ordered loop valued) pseudometric defines a uniformity. In this theory we extend this to the situation where we have an arbitrary collection of pseudometrics, all defined on the the same set X and valued in an ordered loop L. Since real numbers form an ordered loop all results proven in this theory are true for the standard real-valued pseudometrics.

96.1 From collection of pseudometrics to fundamental system of entourages

Suppose $\mathcal{M}$ is a collection of (an ordered loop valued) pseudometrics on X, i.e. $d : X \times X \to L^+$ is a pseudometric for every $d \in \mathcal{M}$. Then, for each $d \in \mathcal{M}$ the sets $\{d^{-1}(\{c \in L^+ : c \leq b\}) : b \in L_+\}$ form a fundamental system of entourages (see `MetricSpace_ZF`).

The next two definitions describe the way a common fundamental system of entourages for $\mathcal{M}$ is constructed. First we take finite subset M of $\mathcal{M}$. Then we choose $f : M \to L_+$. This way for each $d \in M$ the value $f(d)$ is a positive element of L and $\{d^{-1}(\{c \in L^+ : c \leq f(d)\}) : d \in M\}$ is a finite collection of subsets of $X \times X$. Then we take intersections of such finite collections as M varies over $\mathcal{M}$ and f varies over all possible functions mapping M to L_+. To simplify notation for this construction we split it into two steps. In the first step we define a collection of finite intersections resulting from choosing a finite set of pseudometrics M, $f : M \to L_+$ and varying the selector function f over the space of functions mapping M to the set of positive elements of L.

definition
```
UniformGaugeSets(X,L,A,r,M) ≡
{(⋂d∈M. d-({c∈Nonnegative(L,A,r). ⟨c,f(d)⟩ ∈ r})). f∈M→PositiveSet(L,A,r)}
```

In the second step we collect all uniform gauge sets defined above as parameter M vary over all nonempty finite subsets of $\mathcal{M}$. This is the collection of sets that we will show forms a fundamental system of entourages.

definition
```
UniformGauges(X,L,A,r,𝓜) ≡ ⋃M∈FinPow(𝓜)\{∅}. UniformGaugeSets(X,L,A,r,M)
```

The context muliple_pmetric is very similar to the pmetric_space context except that rather than fixing a single pseudometric d we fix a collection of pseudometrics $\mathcal{M}$. That forces the notation for disk, topology, interior and closure to depend on the pseudometric d.

```
locale muliple_pmetric = loop1 +
  fixes 𝓜 and X
  assumes mpmetricAssm: ∀d∈𝓜. IsApseudoMetric(d,X,L,A,r)
  fixes disk
  defines disk_def [simp]: disk(d,c,R) ≡ Disk(X,d,r,c,R)
  fixes pmettop (τ)
  defines pmettop [simp]: τ(d) ≡ MetricTopology(X,L,A,r,d)
  fixes interior (int)
  defines interior_def [simp]: int(d,D) ≡ Interior(D,τ(d))
  fixes cl
  defines cl_def [simp]: cl(d,D) ≡ Closure(D,τ(d))
```

Analogously what is done in the pmetric_space context we will write UniformGauges(X,L,A,r,$\mathcal{M}$) as $\mathfrak{B}$ in the muliple_pmetric context.

```
abbreviation (in muliple_pmetric) mgauge (𝔅) where 𝔅 ≡ UniformGauges(X,L,A,r,𝓜)
```

The next lemma just shows the definition of $\mathfrak{B}$ in notation used in the muliple_pmetric.

```
lemma (in muliple_pmetric) mgauge_def_alt: shows
  𝔅 = (⋃M∈FinPow(𝓜)\{∅}. {(⋂d∈M. d-({c∈L⁺. c≤f(d)})). f∈M→L₊})
  unfolding UniformGaugeSets_def UniformGauges_def by simp
```

$\mathfrak{B}$ consists of (finite) intersections of sets of the form $d^{-1}(\{c \in L^+ : c \leq f(d)\})$ where $f : M \to L_+$ some finite subset $M \subseteq \mathcal{M}$. More precisely, if M is a nonempty finite subset of $\mathcal{M}$ and f maps M to the positive set of the loop L, then the set $\bigcap_{d\in M} d^{-1}(\{c \in L^+ : c \leq f(d)\}$ is in $\mathfrak{B}$.

```
lemma (in muliple_pmetric) mgauge_finset_fun:
  assumes M∈FinPow(𝓜) M≠∅ f:M→L₊
  shows (⋂d∈M. d-({c∈L⁺. c≤f(d)})) ∈ 𝔅
  using assms mgauge_def_alt by auto
```

If d is one of the pseudometrics from $\mathcal{M}$ then theorems proven in pmetric_space can are valid.

```
lemma (in muliple_pmetric) pmetric_space_valid_in_mpm:
  assumes d∈𝓜 shows pmetric_space(L,A,r,d,X)
proof
  from assms mpmetricAssm show IsApseudoMetric(d,X,L,A,r) by simp
qed
```

If d is member of any finite subset of $\mathcal{M}$ then d maps $X \times X$ to the nonnegative set of the ordered loop L.

```
lemma (in muliple_pmetric) each_pmetric_map:
```

```
  assumes M∈FinPow(ℳ) and d∈M shows d: X×X → L⁺
    using assms pmetric_space_valid_in_mpm pmetric_space.pmetric_properties(1)
    unfolding FinPow_def by auto
```

Members of the uniform gauge defined by multiple pseudometrics are subsets of $X \times X$ i.e. relations on X.

```
lemma (in muliple_pmetric) muniform_gauge_relations:
  assumes B∈𝔅 shows B⊆X×X
proof -
  from assms obtain M f where M∈FinPow(ℳ) and
    I: B = (⋂d∈M. d-({c∈L⁺. c≤f(d)}))
    using mgauge_def_alt by auto
  { fix d assume d∈M
    with ‹M∈FinPow(ℳ)› have d: X×X → L⁺
      using each_pmetric_map by simp
    then have d-({c∈L⁺. c≤f(d)}) ⊆ X×X using func1_1_L15 by auto
  } with I show thesis using inter_subsets_subset by simp
qed
```

Suppose M_1 is a subset of M and $\mathcal{M}$. f_1, f map M_1 and M, resp. to L_+ and $f \leq f_1$ on M_1. Then the set $\bigcap_{d\in M} d^{-1}(\{y \in L_+ : y \leq f(d)\})$ is included in the set $\bigcap_{d\in M_1} d^{-1}(\{y \in L_+ : y \leq f_1(d)\})$.

```
lemma (in muliple_pmetric) fun_inter_vimage_mono:
  assumes M₁⊆ℳ M₁⊆M M₁≠∅ f₁:M₁→L₊ f:M→L₊ and
    ∀d∈M₁. f(d)≤f₁(d)
  shows
     (⋂d∈M. d-({c∈L⁺. c≤f(d)})) ⊆ (⋂d∈M₁. d-({c∈L⁺. c≤f₁(d)}))
proof -
  let L = (⋂d∈M. d-({c∈L⁺. c≤f(d)}))
  let R = (⋂d∈M₁. d-({c∈L⁺. c≤f₁(d)}))
  from assms(2,3) have I: L ⊆ (⋂d∈M₁. d-({c∈L⁺. c≤f(d)}))
    using inter_index_mono by simp
  from assms(1,6) have
    ∀d∈M₁. (d-({c∈L⁺. c≤f(d)})) ⊆ d-({c∈L⁺. c≤f₁(d)})
    using pmetric_space_valid_in_mpm pmetric_space.uniform_gauge_mono
      by force
  with assms(2) have  (⋂d∈M₁. d-({c∈L⁺. c≤f(d)})) ⊆ R by force
  with I show L⊆R by (rule subset_trans)
qed
```

For any two sets B_1, B_2 in $\mathfrak{B}$ there exist a third one that is contained in both.

```
lemma (in muliple_pmetric) mgauge_1st_cond:
  assumes r {down-directs} L₊ B₁∈𝔅 B₂∈𝔅
  shows ∃B∈𝔅. B⊆B₁∩B₂
proof -
  from assms(2,3) obtain M₁ f₁ M₂ f₂ where M₁≠∅ M₂≠∅ and
  I: M₁∈FinPow(ℳ) M₂∈FinPow(ℳ) f₁:M₁→L₊ f₂:M₂→L₊ and
```

```
  II: B₁ = (⋂d∈M₁. d-({c∈L⁺. c≤f₁(d)})) B₂ = (⋂d∈M₂. d-({c∈L⁺. c≤f₂(d)}))
    using mgauge_def_alt by auto
  let M₃ = M₁∪M₂
  from assms(1) have IsDownDirectedSet(L₊,r)
    using down_directs_directed by simp
  with I have
    ∃f₃∈M₃→L₊. (∀d∈M₁. ⟨f₃(d),f₁(d)⟩∈r) ∧ (∀d∈M₂. ⟨f₃(d),f₂(d)⟩∈r)
    using two_fun_low_bound by simp
  then obtain f₃ where f₃:M₃→L₊ and
    III: ∀d∈M₁. f₃(d)≤f₁(d) ∀d∈M₂. f₃(d)≤f₂(d)
      by auto
  let B₃ = ⋂d∈M₃. d-({c∈L⁺. c≤f₃(d)})
  from I(1,2) ‹M₁≠∅› ‹f₃:M₃→L₊› have B₃∈𝔅
    using union_finpow mgauge_def_alt by auto
  moreover have B₃⊆B₁∩B₂
  proof -
    from I(1,2) have M₁⊆ℳ M₁⊆M₃ M₂⊆ℳ M₂⊆M₃
      unfolding FinPow_def by auto
    with ‹M₁≠∅› ‹M₂≠∅› I(3,4) II ‹f₃:M₃→L₊› III
    have B₃⊆B₁ and B₃⊆B₂ using fun_inter_vimage_mono by simp_all
    thus B₃⊆B₁∩B₂ by auto
  qed
  ultimately show ∃B∈𝔅. B⊆B₁∩B₂ by (rule witness_exists)
qed
```

Sets in $\mathfrak{B}$ contain the diagonal and are symmetric, hence contain a symmetric subset from $\mathfrak{B}$.

```
lemma (in muliple_pmetric) mgauge_2nd_and_3rd_cond: assumes B∈𝔅
  shows id(X)⊆B B = converse(B) ∃B₂∈𝔅. B₂ ⊆ converse(B)
proof -
  from assms obtain M f where M∈FinPow(ℳ) M≠∅ f:M→L₊ and
    I: B = (⋂d∈M. d-({c∈L⁺. c≤f(d)}))
    using mgauge_def_alt by auto
  { fix d assume d∈M
    let B_d = d-({c∈L⁺. c≤f(d)})
    from ‹d∈M› ‹f:M→L₊› ‹M∈FinPow(ℳ)› have
      pmetric_space(L,A,r,d,X) and B_d ∈ UniformGauge(X,L,A,r,d)
      unfolding FinPow_def UniformGauge_def
      using apply_funtype pmetric_space_valid_in_mpm by auto
    then have id(X)⊆B_d and B_d = converse(B_d)
      using pmetric_space.gauge_2nd_cond pmetric_space.gauge_symmetric

      by simp_all
  } with I ‹M≠∅› show id(X)⊆B and B = converse(B) by auto
  from assms ‹B = converse(B)› show ∃B₂∈𝔅. B₂ ⊆ converse(B)
    by auto
qed
```

$\mathfrak{B}$ is a subset of the power set of $X \times X$.

```
lemma (in muliple_pmetric) mgauge_5thCond: shows 𝔅⊆Pow(X×X)
  using muniform_gauge_relations by auto
```

If $\mathcal{M}$ and L_+ are nonempty then $\mathfrak{B}$ is also nonempty.

```
lemma (in muliple_pmetric) mgauge_6thCond:
  assumes ℳ≠∅ and L₊≠∅ shows 𝔅≠∅
proof -
  from assms obtain M y where M∈FinPow(ℳ) M≠∅ and y∈L₊
    using finpow_nempty_nempty by blast
  from ‹y∈L₊› have ConstantFunction(M,y):M→L₊ using func1_3_L1 by simp
  with ‹M∈FinPow(ℳ)› ‹M≠∅› show 𝔅≠∅ using mgauge_finset_fun by auto
qed
```

If the loop order is halfable then for every set $B_1 \in \mathfrak{B}$ there is another set $B_2 \in \mathfrak{B}$ such that $B_2 \circ B_2 \subseteq B_1$.

```
lemma (in muliple_pmetric) mgauge_4thCond:
  assumes IsHalfable(L,A,r) B₁∈𝔅 shows ∃B₂∈𝔅. B₂ O B₂ ⊆ B₁
proof -
  from assms(2) obtain M f₁ where M∈FinPow(ℳ) M≠∅ f₁:M→L₊ and
    I: B₁ = (⋂d∈M. d-({c∈L⁺. c≤f₁(d)}))
    using mgauge_def_alt by auto
  from assms(1) ‹f₁:M→L₊› have ∀d∈M. ∃b₂∈L₊. b₂+b₂ ≤ f₁(d)
    using apply_funtype unfolding IsHalfable_def by simp
  with ‹M∈FinPow(ℳ)› have ∃f₂∈M→L₊. ∀d∈M. f₂(d) + f₂(d) ≤ f₁(d)
    unfolding FinPow_def using finite_choice_fun by auto
  then obtain f₂ where f₂∈M→L₊ and II: ∀d∈M. f₂(d) + f₂(d) ≤ f₁(d)
    by auto
  let B₂ = ⋂d∈M. d-({c∈L⁺. c≤f₂(d)})
  { fix d assume d∈M
    let A₂ = d-({c∈L⁺. c≤f₂(d)})
    from ‹d∈M› ‹M∈FinPow(ℳ)› have pmetric_space(L,A,r,d,X)
      unfolding FinPow_def using pmetric_space_valid_in_mpm by auto
    with ‹f₂∈M→L₊› ‹d∈M› II have A₂ O A₂ ⊆ d-({c∈L⁺. c≤f₁(d)})
      using apply_funtype pmetric_space.half_vimage_square by simp
  }
  with ‹M≠∅› I have B₂ O B₂ ⊆ B₁ using square_incl_product by simp
  with ‹M∈FinPow(ℳ)› ‹M≠∅› ‹f₂∈M→L₊› show thesis
    using mgauge_finset_fun by auto
qed
```

If $\mathcal{M}$ is a nonempty collection of pseudometrics on a nonempty set X valued in loop L partially ordered by relation r such that the set of positive elements L_+ is nonempty, r down directs L_+ and r is halfable on L,then $\mathfrak{B}$ is a fundamental system of entourages in X hence its supersets form a uniformity on X and hence those supersets define a topology on X

```
lemma (in muliple_pmetric) mmetric_gauge_base:
  assumes X≠∅ ℳ≠∅ L₊≠∅ r {down-directs} L₊ IsHalfable(L,A,r)
  shows
```

```
    𝔅 {is a uniform base on} X
    Supersets(X×X,𝔅) {is a uniformity on} X
    UniformTopology(Supersets(X×X,𝔅),X) {is a topology}
    ⋃UniformTopology(Supersets(X×X,𝔅),X) = X
  using assms mgauge_1st_cond mgauge_2nd_and_3rd_cond mgauge_4thCond
    mgauge_5thCond mgauge_6thCond uniformity_base_is_base uniform_top_is_top
  unfolding IsUniformityBaseOn_def by simp_all
```

96.2 An alternative approach

The formula for defining the fundamental system of entourages from a collection of pseudometrics given in lemma `mgauge_def_alt` is a bit different than the standard one found in the literature on real-valued pseudometrics. In this section we explore another alternative to defining fundamental system of entourages common to a collection of pseudometrics.

Any pseudometrics $d : X \times X \to L^+$ defines a fundamental system of entourages on X by the formula $\mathcal{B}(d) = \{d^{-1}(\{c \in L^+ : c \leq b\}) : b \in L_+\}$ (see theorem `metric_gauge_base` in `Metric_Space_ZF` theory.

```
definition (in muliple_pmetric) gauge (ℬ) where
  ℬ(d) ≡ {d-({c∈L⁺. c≤b}). b∈L₊}
```

Every subset M of $\mathcal{M}$ defines a collection of fundamental systems of entourages $\mathfrak{M}(M) = \{\mathcal{B}(d) : d \in M\}$.

```
definition (in muliple_pmetric) gauge_set (𝔐) where
  𝔐(M) = {ℬ(d). d∈M}
```

To get a fundamental system of entourages common to all pseudometrics $d \in \mathcal{M}$ we take intersections of sets selected from finite nonempty subcollections of the collection of all fundamental systems of entourages defined by pseudometrics $d \in \mathcal{M}$. To distinguish it from the common fundamental system of entourages defined in the previous section we denote that $\mathfrak{B}_1$.

```
definition (in muliple_pmetric) mgauge_alt (𝔅₁) where
  𝔅₁ ≡ ⋃M∈FinPow(ℳ)\{∅}. {(⋂B∈𝔐(M). g(B)). g∈ChoiceFunctions(𝔐(M))}
```

If M is a nonempty finite subset of $mathcalM$ then we have inclusion $\{\bigcap_{B\in\mathfrak{M}(M)} g(B) : g \in (C)(\mathfrak{M}(M)\} \subseteq \{\bigcap_{d\in M} d^{-1}(\{c \in L^+ : c \leq f(d)\} : f : M \to L_+\})$. where $\mathcal{C}(\mathcal{A})$ is the set of choice functions for a collection $\mathcal{A}$ (see theory `Cardinal_ZF` for definition of choice function for a collection.

```
lemma (in muliple_pmetric) mgauge_alt_mgauge1:
  assumes M∈FinPow(ℳ) M≠∅
  defines 𝒞 ≡ ChoiceFunctions(𝔐(M))
  shows {(⋂B∈𝔐(M). g(B)). g∈𝒞} ⊆ {(⋂d∈M. d-({c∈L⁺. c≤f(d)})). f∈M→L₊}
proof -
  let L = {(⋂B∈𝔐(M). g(B)). g∈𝒞}
  let R = {(⋂d∈M. d-({c∈L⁺. c≤f(d)})). f∈M→L₊}
```

```
from assms(1,2) have Finite(𝔐(M)) and 𝔐(M)≠∅
  unfolding gauge_set_def FinPow_def using fin_rep_fin by simp_all
{ fix U assume U ∈ L
  then obtain g where g∈𝒞 and U=(⋂B∈𝔐(M). g(B)) by auto
  from assms(3) ‹g∈𝒞› have g:(𝔐(M))→(⋃𝔐(M)) and
    I: ∀B∈𝔐(M). g(B) ∈ B
    unfolding ChoiceFunctions_def by auto
  { fix d assume d∈M
    then have ℬ(d) ∈ 𝔐(M) and II: ℬ(d) = {d-({c∈L⁺. c≤b}). b∈L₊}
      unfolding gauge_set_def gauge_def by auto
    from I ‹ℬ(d) ∈ 𝔐(M)› have g(ℬ(d)) ∈ ℬ(d) by simp
    with II obtain b where
      b∈L₊ and g(ℬ(d)) = d-({c∈L⁺. c≤b})
      by auto
    hence ∃b∈L₊. g(ℬ(d)) = d-({c∈L⁺. c≤b}) by auto
  } hence ∀d∈M. ∃b∈L₊. g(ℬ(d)) = d-({c∈L⁺. c≤b})
      by auto
  with assms(1) have
    ∃f∈M→L₊. ∀d∈M. g(ℬ(d)) = d-({c∈L⁺. c≤f(d)})
    unfolding FinPow_def using finite_choice_fun by auto
  then obtain f where
    f:M→L₊ and II: ∀d∈M. g(ℬ(d)) = d-({c∈L⁺. c≤f(d)})
    by auto
  have U = (⋂d∈M. d-({c∈L⁺. c≤f(d)}))
  proof
    { fix p assume p∈U
      with ‹U=(⋂B∈𝔐(M). g(B))› have ∀B∈𝔐(M). p∈g(B) by auto
      { fix d assume d∈M
        then have ℬ(d) ∈ 𝔐(M) unfolding gauge_set_def by auto
        with ‹∀B∈𝔐(M). p∈g(B)› have p∈g(ℬ(d)) by simp
        with II ‹d∈M› have p ∈ d-({c∈L⁺. c≤f(d)})
          by simp
      } hence ∀d∈M. p ∈ d-({c∈L⁺. c≤f(d)}) by simp
      with assms(2) have p∈(⋂d∈M. d-({c∈L⁺. c≤f(d)}))
          by auto
    } thus U ⊆ (⋂d∈M. d-({c∈L⁺. c≤f(d)})) by auto
    { fix p assume p ∈ (⋂d∈M. d-({c∈L⁺. c≤f(d)}))
      hence III: ∀d∈M. p ∈ d-({c∈L⁺. c≤f(d)}) by auto
      { fix B assume B∈𝔐(M)
        then have B ∈ {ℬ(d). d∈M} unfolding gauge_set_def
          by simp
        then obtain d where d∈M and B=ℬ(d)
          unfolding gauge_def by auto
        from ‹d∈M› II have g(ℬ(d)) = d-({c∈L⁺. c≤f(d)})
          by simp
        with III ‹d∈M› have p ∈ g(ℬ(d)) by simp
        with ‹B=ℬ(d)› have p∈g(B) by simp
      } hence ∀B∈𝔐(M). p∈g(B) by simp
      with ‹𝔐(M)≠∅› ‹U=(⋂B∈𝔐(M). g(B))› have p∈U by auto
```

```
    } thus (⋂d∈M. d-({c∈L⁺. c≤f(d)})) ⊆ U by auto
  qed
  with ‹f:M→L₊› have U∈R by auto
} thus L⊆R by auto
qed
end
```

97 Topological groups - introduction

```
theory TopologicalGroup_ZF imports Topology_ZF_3 Group_ZF_1 Semigroup_ZF
```

```
begin
```

This theory is about the first subject of algebraic topology: topological groups.

97.1 Topological group: definition and notation

Topological group is a group that is a topological space at the same time. This means that a topological group is a triple of sets, say (G, f, T) such that T is a topology on G, f is a group operation on G and both f and the operation of taking inverse in G are continuous. Since IsarMathLib defines topology without using the carrier, (see `Topology_ZF`), in our setup we just use $\bigcup T$ instead of G and say that the pair of sets $(\bigcup T, f)$ is a group. This way our definition of being a topological group is a statement about two sets: the topology T and the group operation f on $G = \bigcup T$. Since the domain of the group operation is $G \times G$, the pair of topologies in which f is supposed to be continuous is T and the product topology on $G \times G$ (which we will call τ below).

This way we arrive at the following definition of a predicate that states that pair of sets is a topological group.

```
definition
  IsAtopologicalGroup(T,f) ≡ (T {is a topology}) ∧ IsAgroup(⋃T,f) ∧
  IsContinuous(ProductTopology(T,T),T,f) ∧
  IsContinuous(T,T,GroupInv(⋃T,f))
```

We will inherit notation from the `topology0` locale. That locale assumes that T is a topology. For convenience we will denote $G = \bigcup T$ and τ to be the product topology on $G \times G$. To that we add some notation specific to groups. We will use additive notation for the group operation, even though we don't assume that the group is abelian. The notation $g+A$ will mean the left translation of the set A by element g, i.e. $g + A = \{g + a | a \in A\}$. The group operation G induces a natural operation on the subsets of G defined as $\langle A, B\rangle \mapsto \{x + y | x \in A, y \in B\}$. Such operation has been considered in `func_ZF` and called f "lifted to subsets of" G. We will denote the value of such operation on sets A, B as $A + B$. The set of neigboorhoods of zero (denoted $\mathcal{N}_0$) is the collection of (not necessarily open) sets whose interior contains the neutral element of the group.

locale topgroup = topology0 +

```
  fixes G
  defines G_def [simp]: G ≡ ⋃T

  fixes prodtop (τ)
  defines prodtop_def [simp]: τ ≡ ProductTopology(T,T)

  fixes f

  assumes Ggroup: IsAgroup(G,f)

  assumes fcon: IsContinuous(τ,T,f)

  assumes inv_cont: IsContinuous(T,T,GroupInv(G,f))

  fixes grop (infixl + 90)
  defines grop_def [simp]: x+y ≡ f⟨x,y⟩

  fixes grinv (- _ 89)
  defines grinv_def [simp]: (-x) ≡ GroupInv(G,f)(x)

  fixes grsub (infixl - 90)
  defines grsub_def [simp]: x-y ≡ x+(-y)

  fixes setinv (- _ 72)
  defines setninv_def [simp]: -A ≡ GroupInv(G,f)(A)

  fixes ltrans (infix + 73)
  defines ltrans_def [simp]: x + A ≡ LeftTranslation(G,f,x)(A)

  fixes rtrans (infix + 73)
  defines rtrans_def [simp]: A + x ≡ RightTranslation(G,f,x)(A)

  fixes setadd (infixl + 71)
  defines setadd_def [simp]: A+B ≡ (f {lifted to subsets of} G)⟨A,B⟩

  fixes gzero (0)
  defines gzero_def [simp]: 0 ≡ TheNeutralElement(G,f)

  fixes zerohoods (𝒩₀)
  defines zerohoods_def [simp]: 𝒩₀ ≡ {A ∈ Pow(G). 0 ∈ int(A)}

  fixes listsum (∑ _ 70)
  defines listsum_def[simp]: ∑s ≡ Fold(f,0,s)
```

```
  fixes nat_mult (infix · 95)
  defines nat_mult_def [simp]: n·x ≡ ∑{⟨k,x⟩. k∈n}
```

The first lemma states that we indeeed talk about topological group in the context of topgroup locale.

```
lemma (in topgroup) topGroup: shows IsAtopologicalGroup(T,f)
  using topSpaceAssum Ggroup fcon inv_cont IsAtopologicalGroup_def
  by simp
```

If a pair of sets (T, f) forms a topological group, then all theorems proven in the topgroup context are valid as applied to (T, f).

```
lemma topGroupLocale: assumes IsAtopologicalGroup(T,f)
  shows topgroup(T,f)
  using assms IsAtopologicalGroup_def topgroup_def
    topgroup_axioms.intro topology0_def by simp
```

We can use the group0 locale in the context of topgroup.

```
lemma (in topgroup) group0_valid_in_tgroup: shows group0(G,f)
  using Ggroup group0_def by simp
```

We can use the group0 locale in the context of topgroup.

```
sublocale topgroup < group0 G f gzero grop grinv listsum nat_mult
    unfolding group0_def gzero_def grop_def grinv_def using Ggroup by
auto
```

We can use semigr0 locale in the context of topgroup.

```
lemma (in topgroup) semigr0_valid_in_tgroup: shows semigr0(G,f)
  using Ggroup IsAgroup_def IsAmonoid_def semigr0_def by simp
```

We can use the prod_top_spaces0 locale in the context of topgroup.

```
lemma (in topgroup) prod_top_spaces0_valid: shows prod_top_spaces0(T,T,T)
  using topSpaceAssum prod_top_spaces0_def by simp
```

Negative of a group element is in group.

```
lemma (in topgroup) neg_in_tgroup: assumes g∈G shows (-g) ∈ G
  using assms inverse_in_group by simp
```

Sum of two group elements is in the group.

```
lemma (in topgroup) group_op_closed_add:  assumes x₁ ∈ G  x₂ ∈ G
  shows x₁+x₂ ∈ G
  using assms group_op_closed by simp
```

Zero is in the group.

```
lemma (in topgroup) zero_in_tgroup: shows 0∈G
  using group0_2_L2 by simp
```

Another lemma about canceling with two group elements written in additive notation

```
lemma (in topgroup) inv_cancel_two_add:
  assumes x₁ ∈ G  x₂ ∈ G
  shows
    x₁+(-x₂)+x₂ = x₁
    x₁+x₂+(-x₂) = x₁
    (-x₁)+(x₁+x₂) = x₂
    x₁+((-x₁)+x₂) = x₂
  using assms inv_cancel_two by auto
```

Useful identities proven in the Group_ZF theory, rewritten here in additive notation. Note since the group operation notation is left associative we don't really need the first set of parentheses in some cases.

```
lemma (in topgroup) cancel_middle_add: assumes x₁ ∈ G  x₂ ∈ G  x₃ ∈
G
  shows
    (x₁+(-x₂))+(x₂+(-x₃)) = x₁+ (-x₃)
    ((-x₁)+x₂)+((-x₂)+x₃) = (-x₁)+ x₃
    (- (x₁+x₂)) + (x₁+x₃) = (-x₂)+x₃
    (x₁+x₂) + (-(x₃+x₂)) =x₁+ (-x₃)
    (-x₁) + (x₁+x₂+x₃) + (-x₃) = x₂
proof -
  from assms have x₁+ (-x₃) = (x₁+(-x₂))+(x₂+(-x₃))
    using group0_2_L14A(1) by blast
  thus (x₁+(-x₂))+(x₂+(-x₃)) = x₁+ (-x₃) by simp
  from assms have (-x₁)+ x₃ = ((-x₁)+x₂)+((-x₂)+x₃)
    using group0_2_L14A(2) by blast
  thus ((-x₁)+x₂)+((-x₂)+x₃) = (-x₁)+ x₃ by simp
  from assms show (- (x₁+x₂)) + (x₁+x₃) = (-x₂)+x₃
    using cancel_middle(1) by simp
  from assms show (x₁+x₂) + (-(x₃+x₂)) =x₁+ (-x₃)
    using cancel_middle(2) by simp
  from assms show (-x₁) + (x₁+x₂+x₃) + (-x₃) = x₂
    using cancel_middle(3) by simp
qed
```

We can cancel an element on the right from both sides of an equation.

```
lemma (in topgroup) cancel_right_add:
  assumes x₁ ∈ G  x₂ ∈ G  x₃ ∈ G x₁+x₂ = x₃+x₂
  shows x₁ = x₃
  using assms cancel_right by simp
```

We can cancel an element on the left from both sides of an equation.

```
lemma (in topgroup) cancel_left_add:
  assumes x₁ ∈ G  x₂ ∈ G  x₃ ∈ G x₁+x₂ = x₁+x₃
  shows x₂ = x₃
  using assms cancel_left by simp
```

We can put an element on the other side of an equation.

```
lemma (in topgroup) put_on_the_other_side:
  assumes x₁ ∈ G  x₂ ∈ G x₃ = x₁+x₂
  shows x₃+(-x₂) = x₁ and (-x₁)+x₃ = x₂
  using assms group0_2_L18 by auto
```

A simple equation from lemma `simple_equation0` in `Group_ZF` in additive notation

```
lemma (in topgroup) simple_equation0_add:
  assumes x₁ ∈ G  x₂ ∈ G  x₃ ∈ G x₁+(-x₂) = (-x₃)
  shows x₃ = x₂ + (-x₁)
  using assms simple_equation0 by blast
```

A simple equation from lemma `simple_equation1` in `Group_ZF` in additive notation

```
lemma (in topgroup) simple_equation1_add:
  assumes x₁ ∈ G  x₂ ∈ G  x₃ ∈ G (-x₁)+x₂ = (-x₃)
  shows x₃ = (-x₂) + x₁
  using assms simple_equation1 by blast
```

The set comprehension form of negative of a set. The proof uses the `ginv_image` lemma from `Group_ZF` theory which states the same thing in multiplicative notation.

```
lemma (in topgroup) ginv_image_add: assumes V⊆G
  shows (-V)⊆G and (-V) = {-x. x ∈ V}
  using assms ginv_image by auto
```

The additive notation version of `ginv_image_el` lemma from `Group_ZF` theory

```
lemma (in topgroup) ginv_image_el_add: assumes V⊆G x ∈ (-V)
  shows (-x) ∈ V
  using assms ginv_image_el by simp
```

Of course the product topology is a topology (on $G \times G$).

```
lemma (in topgroup) prod_top_on_G:
  shows τ {is a topology} and ⋃τ = G×G
  using topSpaceAssum Top_1_4_T1 by auto
```

Let's recall that f is a binary operation on G in this context.

```
lemma (in topgroup) topgroup_f_binop: shows f : G×G → G
  using Ggroup group0_def group0.group_oper_fun by simp
```

A subgroup of a topological group is a topological group with relative topology and restricted operation. Relative topology is the same as `T {restricted to} H` which is defined to be $\{V \cap H : V \in T\}$ in `ZF1` theory.

```
lemma (in topgroup) top_subgroup: assumes A1: IsAsubgroup(H,f)
  shows IsAtopologicalGroup(T {restricted to} H,restrict(f,H×H))
proof -
  let τ₀ = T {restricted to} H
```

```
  let fH = restrict(f,H×H)
  have ⋃τ0 = G ∩ H using union_restrict by simp
  also from A1 have ... = H
    using group0_3_L2 by blast
  finally have ⋃τ0 = H by simp
  have τ0 {is a topology} using Top_1_L4 by simp
  moreover from A1 ‹⋃τ0 = H› have IsAgroup(⋃τ0,fH)
    using IsAsubgroup_def by simp
  moreover have IsContinuous(ProductTopology(τ0,τ0),τ0,fH)
  proof -
    have two_top_spaces0(τ, T,f)
      using topSpaceAssum prod_top_on_G topgroup_f_binop prod_top_on_G
 two_top_spaces0_def by simp
    moreover
    from A1 have H ⊆ G using group0_3_L2 by simp
    then have H×H ⊆ ⋃τ using prod_top_on_G by auto
    moreover have IsContinuous(τ,T,f) using fcon by simp
    ultimately have
      IsContinuous(τ {restricted to} H×H, T {restricted to} fH(H×H),fH)
using two_top_spaces0.restr_restr_image_cont
      by simp
    moreover have
      ProductTopology(τ0,τ0) = τ {restricted to} H×H using topSpaceAssum
prod_top_restr_comm
      by simp
    moreover from A1 have fH(H×H) = H using image_subgr_op
      by simp
    ultimately show thesis by simp
  qed
  moreover have IsContinuous(τ0,τ0,GroupInv(⋃τ0,fH))
  proof -
    let g = restrict(GroupInv(G,f),H)
    have GroupInv(G,f) : G → G
      using Ggroup group0_2_T2 by simp
    then have two_top_spaces0(T,T,GroupInv(G,f))
      using topSpaceAssum two_top_spaces0_def by simp
    moreover from A1 have H ⊆ ⋃T using group0_3_L2 by simp
    ultimately have
      IsContinuous(τ0,T {restricted to} g(H),g)
      using inv_cont two_top_spaces0.restr_restr_image_cont
      by simp
    moreover from A1 have g(H) = H using restr_inv_onto by simp
    moreover
    from A1 have GroupInv(H,fH) = g using group0_3_T1 by simp
    with ‹⋃τ0 = H› have g = GroupInv(⋃τ0,fH) by simp
    ultimately show thesis by simp
  qed
  ultimately show thesis unfolding IsAtopologicalGroup_def by simp
qed
```

97.2 Interval arithmetic, translations and inverse of set

In this section we list some properties of operations of translating a set and reflecting it around the neutral element of the group. Many of the results are proven in other theories, here we just collect them and rewrite in notation specific to the `topgroup` context.

Different ways of looking at adding sets.

```
lemma (in topgroup) interval_add: assumes A⊆G B⊆G shows
  A+B ⊆ G
  A+B = f(A×B)
  A+B = (⋃x∈A. x+B)
  A+B = {x+y. ⟨x,y⟩ ∈ A×B}
proof -
  from assms show A+B ⊆ G and A+B = f(A×B) and A+B = {x+y. ⟨x,y⟩ ∈
A×B}
    using topgroup_f_binop lift_subsets_explained by auto
  from assms show A+B = (⋃x∈A. x+B) using image_ltrans_union by simp
qed
```

If the neutral element is in a set, then it is in the sum of the sets.

```
lemma (in topgroup) interval_add_zero: assumes A⊆G 0∈A
  shows 0 ∈ A+A
proof -
  from assms have 0+0 ∈ A+A using interval_add(4) by auto
  then show 0 ∈ A+A using group0_2_L2 by auto
qed
```

Some lemmas from `Group_ZF_1` about images of set by translations written in additive notation

```
lemma (in topgroup) lrtrans_image: assumes V⊆G x∈G
  shows
    x+V = {x+v. v∈V}
    V+x = {v+x. v∈V}
  using assms ltrans_image rtrans_image by auto
```

Right and left translations of a set are subsets of the group. This is of course typically applied to the subsets of the group, but formally we don't need to assume that.

```
lemma (in topgroup) lrtrans_in_group_add: assumes x∈G
  shows  x+V ⊆ G and V+x ⊆G
  using assms lrtrans_in_group by auto
```

A corollary from `interval_add`

```
corollary (in topgroup) elements_in_set_sum: assumes A⊆G B⊆G
  t ∈ A+B shows ∃s∈A. ∃q∈B. t=s+q
  using assms interval_add(4) by auto
```

A corollary from `lrtrans_image`

```
corollary (in topgroup) elements_in_ltrans:
  assumes B⊆G g∈G t ∈ g+B
  shows ∃q∈B. t=g+q
  using assms lrtrans_image(1) by simp
```

Another corollary of `lrtrans_image`

```
corollary (in topgroup) elements_in_rtrans:
  assumes B⊆G g∈G  t ∈ B+g shows ∃q∈B. t=q+g
  using assms lrtrans_image(2) by simp
```

Another corollary from `interval_add`

```
corollary (in topgroup) elements_in_set_sum_inv:
  assumes A⊆G B⊆G t=s+q s∈A q∈B
  shows t ∈ A+B
  using assms interval_add by auto
```

Another corollary of `lrtrans_image`

```
corollary (in topgroup) elements_in_ltrans_inv: assumes B⊆G g∈G q∈B t=g+q
  shows t ∈ g+B
  using assms lrtrans_image(1) by auto
```

Another corollary of `rtrans_image_add`

```
lemma (in topgroup) elements_in_rtrans_inv:
  assumes B⊆G g∈G q∈B t=q+g
  shows t ∈ B+g
  using assms lrtrans_image(2) by auto
```

Right and left translations are continuous.

```
lemma (in topgroup) trans_cont: assumes g∈G shows
  IsContinuous(T,T,RightTranslation(G,f,g)) and
  IsContinuous(T,T,LeftTranslation(G,f,g))
using assms trans_eq_section topgroup_f_binop fcon prod_top_spaces0_valid

  prod_top_spaces0.fix_1st_var_cont prod_top_spaces0.fix_2nd_var_cont
  by auto
```

Left and right translations of an open set are open.

```
lemma (in topgroup) open_tr_open: assumes g∈G and V∈T
  shows g+V ∈ T and  V+g ∈ T
  using assms neg_in_tgroup trans_cont IsContinuous_def trans_image_vimage
by auto
```

Right and left translations are homeomorphisms.

```
lemma (in topgroup) tr_homeo: assumes g∈G shows
  IsAhomeomorphism(T,T,RightTranslation(G,f,g)) and
  IsAhomeomorphism(T,T,LeftTranslation(G,f,g))
```

```
  using assms trans_bij trans_cont open_tr_open bij_cont_open_homeo
  by auto
```

Left translations preserve interior.

```
lemma (in topgroup) ltrans_interior: assumes A1: g∈G and A2: A⊆G
  shows g + int(A) = int(g+A)
proof -
  from assms have A ⊆ ⋃T and IsAhomeomorphism(T,T,LeftTranslation(G,f,g))
using tr_homeo
    by auto
  then show thesis using int_top_invariant by simp
qed
```

Right translations preserve interior.

```
lemma (in topgroup) rtrans_interior: assumes A1: g∈G and A2: A⊆G
  shows int(A) + g = int(A+g)
proof -
  from assms have A ⊆ ⋃T and IsAhomeomorphism(T,T,RightTranslation(G,f,g))
using tr_homeo
    by auto
  then show thesis using int_top_invariant by simp
qed
```

Translating by an inverse and then by an element cancels out.

```
lemma (in topgroup) trans_inverse_elem: assumes g∈G and A⊆G
  shows g+((-g)+A) = A
  using assms neg_in_tgroup trans_comp_image group0_2_L6 trans_neutral
image_id_same
  by simp
```

Inverse of an open set is open.

```
lemma (in topgroup) open_inv_open: assumes V∈T shows (-V) ∈ T
  using assms inv_image_vimage inv_cont IsContinuous_def by simp
```

Inverse is a homeomorphism.

```
lemma (in topgroup) inv_homeo: shows IsAhomeomorphism(T,T,GroupInv(G,f))
  using group_inv_bij inv_cont open_inv_open bij_cont_open_homeo by simp
```

Taking negative preserves interior.

```
lemma (in topgroup) int_inv_inv_int: assumes A ⊆ G
  shows int(-A) = -(int(A))
  using assms inv_homeo int_top_invariant by simp
```

97.3 Neighborhoods of zero

Zero neighborhoods are (not necessarily open) sets whose interior contains the neutral element of the group. In the topgroup locale the collection of neighboorhoods of zero is denoted $\mathcal{N}_0$.

The whole space is a neighborhood of zero.

```
lemma (in topgroup) zneigh_not_empty: shows G ∈ 𝒩₀
  using topSpaceAssum IsATopology_def Top_2_L3 zero_in_tgroup
  by simp
```

Any element that belongs to a subset of the group belongs to that subset with the interior of a neighborhood of zero added.

```
lemma (in topgroup) elem_in_int_sad: assumes A⊆G g∈A H ∈ 𝒩₀
  shows g ∈ A+int(H)
proof -
  from assms(3) have 0 ∈ int(H) and int(H) ⊆ G using Top_2_L2 by auto
  with assms(1,2) have g+0 ∈ A+int(H) using elements_in_set_sum_inv
    by simp
  with assms(1,2) show thesis using group0_2_L2 by auto
qed
```

Any element belongs to the interior of any neighboorhood of zero left translated by that element.

```
lemma (in topgroup) elem_in_int_ltrans:
  assumes g∈G and H ∈ 𝒩₀
  shows g ∈ int(g+H) and g ∈ int(g+H) + int(H)
proof -
  from assms(2) have 0 ∈ int(H) and int(H) ⊆ G using Top_2_L2 by auto
  with assms(1) have g ∈ g + int(H) using neut_trans_elem by simp
  with assms show g ∈ int(g+H) using ltrans_interior by simp
  from assms(1) have int(g+H) ⊆ G using lrtrans_in_group_add(1) Top_2_L1
    by blast
  with ‹g ∈ int(g+H)› assms(2) show g ∈ int(g+H) + int(H)
    using elem_in_int_sad by simp
qed
```

Any element belongs to the interior of any neighboorhood of zero right translated by that element.

```
lemma (in topgroup) elem_in_int_rtrans:
  assumes A1: g∈G and A2: H ∈ 𝒩₀
  shows g ∈ int(H+g) and g ∈ int(H+g) + int(H)
proof -
  from A2 have 0 ∈ int(H) and int(H) ⊆ G using Top_2_L2 by auto
  with A1 have g ∈ int(H) + g using neut_trans_elem by simp
  with assms show g ∈ int(H+g) using rtrans_interior by simp
  from assms(1) have int(H+g) ⊆ G using lrtrans_in_group_add(2) Top_2_L1
    by blast
  with ‹g ∈ int(H+g)› assms(2) show g ∈ int(H+g) + int(H)
    using elem_in_int_sad by simp
qed
```

Negative of a neighborhood of zero is a neighborhood of zero.

```
lemma (in topgroup) neg_neigh_neigh: assumes H ∈ 𝒩₀
```

```
  shows (-H) ∈ 𝒩₀
proof -
  from assms have int(H) ⊆ G and 0 ∈ int(H) using Top_2_L1 by auto
  with assms have 0 ∈ int(-H) using neut_inv_neut int_inv_inv_int by
simp
  moreover
  have GroupInv(G,f):G→G using Ggroup group0_2_T2 by simp
  then have (-H) ⊆ G using func1_1_L6 by simp
  ultimately show thesis by simp
qed
```

Left translating an open set by a negative of a point that belongs to it makes it a neighboorhood of zero.

```
lemma (in topgroup) open_trans_neigh: assumes A1: U∈T and g∈U
  shows (-g)+U ∈ 𝒩₀
proof -
  let H = (-g)+U
  from assms have g∈G by auto
  then have (-g) ∈ G using neg_in_tgroup by simp
  with A1 have H∈T using open_tr_open by simp
  hence H ⊆ G by auto
  moreover have 0 ∈ int(H)
  proof -
    from assms have U⊆G and g∈U by auto
    with ‹H∈T› show 0 ∈ int(H) using elem_trans_neut Top_2_L3 by auto
  qed
  ultimately show thesis by simp
qed
```

Right translating an open set by a negative of a point that belongs to it makes it a neighboorhood of zero.

```
lemma (in topgroup) open_trans_neigh_2: assumes A1: U∈T and g∈U
  shows U+(-g) ∈ 𝒩₀
proof -
  let H = U+(-g)
  from assms have g∈G by auto
  then have (-g) ∈ G using neg_in_tgroup by simp
  with A1 have H∈T using open_tr_open by simp
  hence H ⊆ G by auto
  moreover have 0 ∈ int(H)
  proof -
    from assms have U⊆G and g∈U by auto
    with ‹H∈T› show 0 ∈ int(H) using elem_trans_neut Top_2_L3  by auto
  qed
  ultimately show thesis by simp
qed
```

Right and left translating an neighboorhood of zero by a point and its negative makes it back a neighboorhood of zero.

```
lemma (in topgroup) lrtrans_neigh: assumes W∈𝒩₀ and x∈G
  shows x+(W+(-x)) ∈ 𝒩₀ and (x+W)+(-x) ∈ 𝒩₀
proof -
  from assms(2) have x+(W+(-x)) ⊆ G using lrtrans_in_group_add(1) by
simp
  moreover have 0 ∈ int(x+(W+(-x)))
  proof -
    from assms(2) have int(W+(-x)) ⊆ G
      using neg_in_tgroup lrtrans_in_group_add(2) Top_2_L1 by blast
    with assms(2) have (x+int((W+(-x)))) = {x+y. y∈int(W+(-x))}
      using lrtrans_image(1) by simp
    moreover from assms have (-x) ∈ int(W+(-x))
      using neg_in_tgroup elem_in_int_rtrans(1) by simp
    ultimately have x+(-x) ∈ x+int(W+(-x)) by auto
    with assms show thesis using group0_2_L6 neg_in_tgroup lrtrans_in_group_add(2)
ltrans_interior
      by simp
  qed
  ultimately show x+(W+(-x)) ∈ 𝒩₀ by simp
  from assms(2) have (x+W)+(-x) ⊆ G using lrtrans_in_group_add(2) neg_in_tgroup

    by simp
  moreover have 0 ∈ int((x+W)+(-x))
  proof -
    from assms(2) have int((x+W)) ⊆ G using lrtrans_in_group_add(1) Top_2_L1
by blast
    with assms(2) have int(x+W) + (-x) = {y+(-x).y∈int(x+W)}
      using neg_in_tgroup  lrtrans_image(2) by simp
    moreover from assms have x ∈ int(x+W) using elem_in_int_ltrans(1)
by simp
    ultimately have x+(-x) ∈ int(x+W) + (-x) by auto
    with assms(2) have 0 ∈ int(x+W) + (-x) using group0_2_L6 by simp
    with assms show thesis using group0_2_L6 neg_in_tgroup lrtrans_in_group_add(1)
rtrans_interior
      by auto
  qed
  ultimately show (x+W)+(-x) ∈ 𝒩₀ by simp
qed
```

If A is a subset of B translated by $-x$ then its translation by x is a subset of B.

```
lemma (in topgroup) trans_subset:
  assumes A ⊆ ((-x)+B)x∈G B⊆G
  shows x+A ⊆ B
proof-
  from assms(1) have x+A ⊆ (x+ ((-x)+B)) by auto
  with assms(2,3) show x+A ⊆ B
    using neg_in_tgroup trans_comp_image group0_2_L6 trans_neutral image_id_same
by simp
```

qed

Every neighborhood of zero has a symmetric subset that is a neighborhood of zero.

```
theorem (in topgroup) exists_sym_zerohood:
  assumes U∈𝒩₀
  shows ∃V∈𝒩₀. (V⊆U ∧ (-V)=V)
proof
  let V = U∩(-U)
  have U⊆G using assms unfolding zerohoods_def by auto
  then have V⊆G by auto
  have invg: GroupInv(G, f) ∈ G → G using group0_2_T2 Ggroup by auto
  have invb:GroupInv(G, f) ∈bij(G,G) using group_inv_bij(2) by auto
  have (-V)=GroupInv(G,f)-V unfolding setninv_def using inv_image_vimage

    by auto
  also have ...=(GroupInv(G,f)-U)∩(GroupInv(G,f)-(-U)) using invim_inter_inter_invim
invg
    by auto
  also have ...=(-U)∩(GroupInv(G,f)-(GroupInv(G,f)U))
    unfolding setninv_def using inv_image_vimage by auto
  also from ‹U⊆G› have ...=(-U)∩U using inj_vimage_image invb unfold-
ing bij_def
    by auto
  finally have (-V)=V by auto
  then show V ⊆ U ∧ (- V) = V by auto
  from assms have (-U)∈𝒩₀ using neg_neigh_neigh by auto
  with assms have 0 ∈ int(U)∩int(-U) unfolding zerohoods_def by auto
  moreover have int(U)∩int(-U) = int(V) using int_inter_int by simp
  ultimately have 0 ∈ int(V) by (rule set_mem_eq)
  with ‹V⊆G› show V∈𝒩₀ using zerohoods_def by auto
qed
```

We can say even more than in `exists_sym_zerohood`: every neighborhood of zero U has a symmetric subset that is a neighborhood of zero and its set double is contained in U.

```
theorem (in topgroup) exists_procls_zerohood:
  assumes U∈𝒩₀
  shows ∃V∈𝒩₀. (V⊆U∧ (V+V)⊆U ∧ (-V)=V)
proof-
  have int(U)∈T using Top_2_L2 by auto
  then have f-(int(U))∈τ using fcon IsContinuous_def by auto
  moreover have fne:f ⟨0, 0⟩ = 0 using group0_2_L2 by auto
  moreover
  have 0∈int(U) using assms unfolding zerohoods_def by auto
  then have f - {0}⊆f-(int(U)) using func1_1_L8 vimage_def by auto
  then have GroupInv(G,f)⊆f-(int(U)) using group0_2_T3 by auto
  then have ⟨0,0⟩∈f-(int(U)) using fne zero_in_tgroup unfolding GroupInv_def
    by auto
```

```
  ultimately obtain W V where
    wop:W∈T and vop:V∈T and cartsub:W×V⊆f-(int(U)) and zerhood:⟨0,0⟩∈W×V

    using prod_top_point_neighb topSpaceAssum
    unfolding prodtop_def by force
  then have 0∈W and 0∈V by auto
  then have 0∈W∩V by auto
  have sub:W∩V⊆G using wop vop G_def by auto
  have assoc:f∈G×G→G using group_oper_fun by auto
  {
    fix t s assume t∈W∩V and s∈W∩V
    then have t∈W and s∈V by auto
    then have ⟨t,s⟩∈W×V by auto
    then have ⟨t,s⟩∈f-(int(U)) using cartsub by auto
    then have f⟨t,s⟩∈int(U) using func1_1_L15 assoc by auto
  } hence {f⟨t,s⟩. ⟨t,s⟩∈(W∩V)×(W∩V)}⊆int(U) by auto
  then have (W∩V)+(W∩V)⊆int(U)
    unfolding setadd_def using lift_subsets_explained(4) assoc sub
    by auto
  then have (W∩V)+(W∩V)⊆U using Top_2_L1 by auto
  from topSpaceAssum have W∩V∈T using vop wop unfolding IsATopology_def
by auto
  then have int(W∩V)=W∩V using Top_2_L3 by auto
  with sub ‹0∈W∩V› have W∩V∈𝒩0 unfolding zerohoods_def by auto
  then obtain Q where Q∈𝒩0 and Q⊆W∩V and (-Q)=Q using exists_sym_zerohood
by blast
  then have Q×Q⊆(W∩V)×(W∩V) by auto
  moreover from ‹Q⊆W∩V› have W∩V⊆G and Q⊆G using vop wop unfolding
G_def by auto
  ultimately have Q+Q⊆(W∩V)+(W∩V) using interval_add(2) func1_1_L8 by
auto
  with ‹(W∩V)+(W∩V)⊆U› have Q+Q⊆U by auto
  from ‹Q∈𝒩0› have 0∈Q unfolding zerohoods_def using Top_2_L1 by auto
  with ‹Q+Q⊆U› ‹Q⊆G› have 0+Q⊆U using interval_add(3) by auto
  with ‹Q⊆G› have Q⊆U unfolding ltrans_def gzero_def using trans_neutral(2)
image_id_same
    by auto
  with ‹Q∈𝒩0› ‹Q+Q⊆U› ‹(-Q)=Q› show thesis by auto
qed
```

97.4 Closure in topological groups

This section is devoted to a characterization of closure in topological groups.

Closure of a set is contained in the sum of the set and any neighboorhood of zero.

```
lemma (in topgroup) cl_contains_zneigh:
  assumes A1: A⊆G and A2: H ∈ 𝒩0
  shows cl(A) ⊆ A+H
```

```
proof
  fix x assume x ∈ cl(A)
  from A1 have cl(A) ⊆ G using Top_3_L11 by simp
  with ‹x ∈ cl(A)› have x∈G by auto
  have int(H) ⊆ G using Top_2_L2 by auto
  let V = int(x + (-H))
  have V = x + (-int(H))
  proof -
    from A2 ‹x∈G› have V = x + int(-H)
      using neg_neigh_neigh ltrans_interior by simp
    with A2 show thesis  using int_inv_inv_int by simp
  qed
  have A∩V ≠ 0
  proof -
    from A2 ‹x∈G› ‹x ∈ cl(A)› have V∈T and x ∈ cl(A) ∩ V
      using neg_neigh_neigh elem_in_int_ltrans(1) Top_2_L2 by auto
    with A1 show A∩V ≠ 0 using cl_inter_neigh by simp
  qed
  then obtain y where y∈A and y∈V by auto
  with ‹V = x + (-int(H))› ‹int(H) ⊆ G› ‹x∈G› have x ∈ y+int(H)
    using ltrans_inv_in by simp
  with ‹y∈A› have x ∈ (⋃y∈A. y+H) using Top_2_L1 func1_1_L8 by auto
  with assms show x ∈ A+H using interval_add(3) by simp
qed
```

The next theorem provides a characterization of closure in topological groups in terms of neighborhoods of zero.

```
theorem (in topgroup) cl_topgroup:
  assumes A⊆G shows cl(A) = (⋂H∈𝒩₀. A+H)
proof
  from assms show cl(A) ⊆ (⋂H∈𝒩₀. A+H)
    using zneigh_not_empty cl_contains_zneigh by auto
next
  { fix x assume x ∈ (⋂H∈𝒩₀. A+H)
    then have x ∈ A+G using zneigh_not_empty by auto
    with assms have x∈G using interval_add by blast
    have ∀U∈T. x∈U ⟶ U∩A ≠ 0
    proof -
      { fix U assume U∈T and x∈U
        let H = -((-x)+U)
        from ‹U∈T› and ‹x∈U› have (-x)+U ⊆ G and H ∈ 𝒩₀
          using open_trans_neigh neg_neigh_neigh by auto
        with ‹x ∈ (⋂H∈𝒩₀. A+H)› have x ∈ A+H by auto
        with assms ‹H ∈ 𝒩₀› obtain y where y∈A and x ∈ y+H
          using interval_add(3) by auto
        have y∈U
        proof -
          from assms ‹y∈A› have y∈G by auto
          with ‹(-x)+U ⊆ G› and ‹x ∈ y+H› have y ∈ x+((-x)+U)
```

```
            using ltrans_inv_in by simp
          with ‹U∈T› ‹x∈G› show y∈U
            using neg_in_tgroup trans_comp_image group0_2_L6 trans_neutral
image_id_same
              by auto
        qed
        with ‹y∈A› have U∩A ≠ 0 by auto
      } thus thesis by simp
    qed
    with assms ‹x∈G› have x ∈ cl(A) using inter_neigh_cl by simp
  } thus (⋂H∈𝒩₀. A+H) ⊆ cl(A) by auto
qed
```

97.5 Sums of sequences of elements and subsets

In this section we consider properties of the function $G^n \to G, x = (x_0, x_1, ..., x_{n-1}) \mapsto \sum_{i=0}^{n-1} x_i$. We will model the cartesian product G^n by the space of sequences $n \to G$, where $n = \{0, 1, ..., n-1\}$ is a natural number. This space is equipped with a natural product topology defined in Topology_ZF_3.

Let's recall first that the sum of elements of a group is an element of the group.

```
lemma (in topgroup) sum_list_in_group:
  assumes n ∈ nat and x: succ(n)→G
  shows (∑x) ∈ G
  using assms list_prod_in_group by blast
```

In this context x+y is the same as the value of the group operation on the elements x and y. Normally we shouldn't need to state this a s separate lemma.

```
lemma (in topgroup) grop_def1: shows f⟨x,y⟩ = x+y by simp
```

Another theorem from Semigroup_ZF theory that is useful to have in the additive notation.

```
lemma (in topgroup) shorter_set_add:
  assumes n ∈ nat and x: succ(succ(n))→G
  shows (∑x) = (∑Init(x)) + (x(succ(n)))
proof -
  from assms have semigr0(G,f) and n ∈ nat x: succ(succ(n))→G
    using semigr0_valid_in_tgroup by auto
  then have Fold1(f,x) = f⟨Fold1(f,Init(x)),x(succ(n))⟩
    by (rule semigr0.shorter_seq)
  moreover from assms have (∑x) = Fold1(f,x)
    using nempty_list_prod_as_fold1 by auto
  moreover from assms have (∑Init(x)) = Fold1(f,Init(x))
    using init_props(1) nempty_list_prod_as_fold1 by simp
  ultimately show thesis by simp
qed
```

Sum is a continuous function in the product topology.

```
theorem (in topgroup) sum_continuous: assumes n ∈ nat
  shows IsContinuous(SeqProductTopology(succ(n),T),T,{⟨x,∑x⟩.x∈succ(n)→G})
  proof -
    note ‹n ∈ nat›
    moreover have IsContinuous(SeqProductTopology(succ(0),T),T,{⟨x,∑x⟩.x∈succ(0)→G})
    proof -
      have {⟨x,∑x⟩. x∈succ(0)→G} = {⟨x,x(0)⟩. x∈1→G}
        using prod_singleton by simp
      moreover have
        IsAhomeomorphism(SeqProductTopology(1,T),T,{⟨x,x(0)⟩. x∈1→⋃T})
using topSpaceAssum singleton_prod_top1
          by simp
      ultimately show thesis using IsAhomeomorphism_def by simp
    qed
    moreover have ∀k∈nat.
      IsContinuous(SeqProductTopology(succ(k),T),T,{⟨x,∑x⟩.x∈succ(k)→G})
      ⟶
      IsContinuous(SeqProductTopology(succ(succ(k)),T),T,{⟨x,∑x⟩.x∈succ(succ(k))→G})
      proof -
        { fix k assume k ∈ nat
          let s = {⟨x,∑x⟩.x∈succ(k)→G}
          let g = {⟨p,⟨s(fst(p)),snd(p)⟩⟩. p ∈ (succ(k)→G)×G}
          let h = {⟨x,⟨Init(x),x(succ(k))⟩⟩. x ∈ succ(succ(k))→G}
          let φ = SeqProductTopology(succ(k),T)
          let ψ = SeqProductTopology(succ(succ(k)),T)
          assume IsContinuous(φ,T,s)
          from ‹k ∈ nat› have s: (succ(k)→G) → G
            using sum_list_in_group ZF_fun_from_total by simp
          have h: (succ(succ(k))→G)→(succ(k)→G)×G
          proof -
            { fix x assume x ∈ succ(succ(k))→G
              with ‹k ∈ nat› have Init(x) ∈ (succ(k)→G)
                using init_props by simp
              with ‹k ∈ nat› ‹x : succ(succ(k))→G›
                have ⟨Init(x),x(succ(k))⟩ ∈ (succ(k)→G)×G using apply_funtype
                  by blast
            } then show thesis using ZF_fun_from_total by simp
          qed
          moreover have g:((succ(k)→G)×G)→(G×G)
          proof -
            { fix p assume p ∈ (succ(k)→G)×G
              hence fst(p): succ(k)→G and snd(p) ∈ G by auto
              with ‹s: (succ(k)→G) → G› have ⟨s(fst(p)),snd(p)⟩ ∈ G×G
                using apply_funtype by blast
            } then show g:((succ(k)→G)×G)→(G×G) using ZF_fun_from_total
              by simp
          qed
          moreover have f : G×G → G using topgroup_f_binop by simp
```

```
ultimately have f O g O h :(succ(succ(k))→G)→G using comp_fun
  by blast
from ‹k ∈ nat› have IsContinuous(ψ,ProductTopology(φ,T),h)
  using topSpaceAssum finite_top_prod_homeo IsAhomeomorphism_def
  by simp
moreover have IsContinuous(ProductTopology(φ,T),τ,g)
proof -
  from topSpaceAssum have
     T {is a topology} φ {is a topology} ⋃φ = succ(k)→G
     using seq_prod_top_is_top by auto
  moreover from ‹⋃φ = succ(k)→G› ‹s: (succ(k)→G) → G›
    have s:⋃φ→⋃T by simp
  moreover note ‹IsContinuous(φ,T,s)›
  moreover from ‹⋃φ = succ(k)→G›
    have g = {⟨p,⟨s(fst(p)),snd(p)⟩⟩. p ∈ ⋃φ×⋃T}
    by simp
  ultimately have IsContinuous(ProductTopology(φ,T),ProductTopology(T,T),g)
    using cart_prod_cont1 by blast
  thus thesis by simp
qed
moreover have IsContinuous(τ,T,f) using fcon by simp
moreover have {⟨x,∑x⟩.x∈succ(succ(k))→G} = f O g O h
proof -
  let d = {⟨x,∑x⟩.x∈succ(succ(k))→G}
  from ‹k∈nat› have ∀x∈succ(succ(k))→G. (∑x) ∈ G
    using sum_list_in_group by blast
  then have d:(succ(succ(k))→G)→G
    using sum_list_in_group ZF_fun_from_total by simp
  moreover note ‹f O g O h :(succ(succ(k))→G)→G›
  moreover have ∀x∈succ(succ(k))→G. d(x) = (f O g O h)(x)
  proof
    fix x assume x∈succ(succ(k))→G
    then have I: h(x) = ⟨Init(x),x(succ(k))⟩
      using ZF_fun_from_tot_val1 by simp
    moreover from ‹k∈nat› ‹x∈succ(succ(k))→G›
      have Init(x): succ(k)→G
      using init_props by simp
    moreover from ‹k∈nat› ‹x:succ(succ(k))→G›
      have II: x(succ(k)) ∈ G
      using apply_funtype by blast
    ultimately have h(x) ∈ (succ(k)→G)×G by simp
    then have g(h(x)) = ⟨s(fst(h(x))),snd(h(x))⟩
      using ZF_fun_from_tot_val1 by simp
    with I have g(h(x)) = ⟨s(Init(x)),x(succ(k))⟩
      by simp
    with ‹Init(x): succ(k)→G› have g(h(x)) = ⟨∑Init(x),x(succ(k))⟩
      using ZF_fun_from_tot_val1 by simp
    with ‹k ∈ nat› ‹x: succ(succ(k))→G›
      have f(g(h(x))) = (∑x)
```

```
            using shorter_set_add by simp
          with ‹x ∈ succ(succ(k))→G› have f(g(h(x))) = d(x)
            using ZF_fun_from_tot_val1 by simp
          moreover from
            ‹h: (succ(succ(k))→G)→(succ(k)→G)×G›
            ‹g:((succ(k)→G)×G)→(G×G)›
            ‹f:(G×G)→G› ‹x∈succ(succ(k))→G›
            have (f O g O h)(x) = f(g(h(x))) by (rule func1_1_L18)
          ultimately show d(x) = (f O g O h)(x) by simp
        qed
        ultimately show {⟨x,∑ x⟩.x∈succ(succ(k))→G} = f O g O h
          using func_eq by simp
      qed
      moreover note ‹IsContinuous(τ,T,f)›
      ultimately have IsContinuous(ψ,T,{⟨x,∑ x⟩.x∈succ(succ(k))→G})
        using comp_cont3 by simp
    } thus thesis by simp
  qed
  ultimately show thesis by (rule ind_on_nat)
qed
end
```

98 Topological groups 1

```
theory TopologicalGroup_ZF_1 imports TopologicalGroup_ZF Topology_ZF_properties_2
begin
```

This theory deals with some topological properties of topological groups.

98.1 Separation properties of topological groups

The topological groups have very specific properties. For instance, G is T_0 iff it is T_3.

```
theorem(in topgroup) cl_point:
  assumes x∈G
  shows cl({x}) = (⋂H∈𝒩₀. x+H)
proof-
  {
    have c:cl({x}) = (⋂H∈𝒩₀. {x}+H) using cl_topgroup assms by auto
    {
      fix H
      assume H∈𝒩₀
      then have {x}+H=x+ H using interval_add(3) assms
        by auto
      with ‹H∈𝒩₀› have {x}+H∈{x+H. H∈𝒩₀} by auto
    }
    then have {{x}+H. H∈𝒩₀}⊆{x+H. H∈𝒩₀} by auto
    moreover
```

```
    {
      fix H
      assume H∈𝒩₀
      then have {x}+H=x+ H using interval_add(3) assms
        by auto
      with ‹H∈𝒩₀› have x+ H∈{{x}+H. H∈𝒩₀} by auto
    }
    then have {x+H. H∈𝒩₀}⊆{{x}+H. H∈𝒩₀} by auto
    ultimately have {{x}+H. H∈𝒩₀}={x+H. H∈𝒩₀} by auto
    then have (⋂H∈𝒩₀. {x}+H) = (⋂H∈𝒩₀. x+H) by auto
    with c show cl({x})=(⋂H∈𝒩₀. x+H) by auto
  }
qed
```

We prove the equivalence between T_0 and T_1 first.

```
theorem (in topgroup) neu_closed_imp_T1:
  assumes {0}{is closed in}T
  shows T{is T₁}
proof-
  {
    fix x z assume xG:x∈G and zG:z∈G and dis:x≠z
    then have clx:cl({x})=(⋂H∈𝒩₀. x+H) using cl_point by auto
    {
      fix y
      assume y∈cl({x})
      with clx have y∈(⋂H∈𝒩₀. x+H) by auto
      then have t:∀H∈𝒩₀. y∈x+H by auto
      from ‹y∈cl({x})› xG have yG:y∈G using Top_3_L11(1) G_def by auto
      {
        fix H
        assume HNeig:H∈𝒩₀
        with t have y∈x+H by auto
        then obtain n where y=x+n and n∈H unfolding ltrans_def grop_def
LeftTranslation_def by auto
        with HNeig have nG:n∈G unfolding zerohoods_def by auto
        from ‹y=x+n› and ‹n∈H› have (-x)+y∈H using group0.group0_2_L18(2)
group0_valid_in_tgroup xG nG yG unfolding grinv_def grop_def
          by auto
      }
      then have el:(-x)+y∈(⋂𝒩₀) using zneigh_not_empty by auto
      have cl({0})=(⋂H∈𝒩₀. 0+H) using cl_point zero_in_tgroup by auto
      moreover
      {
        fix H  assume H∈𝒩₀
        then have H⊆G unfolding zerohoods_def by auto
        then have 0+H=H using image_id_same  group0.trans_neutral(2)
group0_valid_in_tgroup unfolding gzero_def ltrans_def
          by auto
        with ‹H∈𝒩₀› have 0+H∈𝒩₀ H∈{0+H. H∈𝒩₀} by auto
```

```
      }
      then have {0+H. H∈𝒩₀}=𝒩₀ by blast
      ultimately have cl({0})=(⋂𝒩₀) by auto
      with el have (-x)+y∈cl({0}) by auto
      then have (-x)+y∈{0} using assms Top_3_L8 G_def zero_in_tgroup
by auto
      then have (-x)+y=0 by auto
      then have y=-(-x) using group0.group0_2_L9(2) group0_valid_in_tgroup
neg_in_tgroup xG yG unfolding grop_def grinv_def by auto
      then have y=x using group0.group_inv_of_inv group0_valid_in_tgroup
xG unfolding grinv_def by auto
    }
    then have cl({x})⊆{x} by auto
    then have cl({x})={x} using xG cl_contains_set G_def by blast
    then have {x}{is closed in}T using Top_3_L8 xG G_def by auto
    then have (⋃T)-{x}∈T using IsClosed_def by auto moreover
    from dis zG G_def have z∈((⋃T)-{x}) ∧ x∉((⋃T)-{x}) by auto
    ultimately have ∃V∈T. z∈V∧x∉V by(safe,auto)
  }
  then show T{is T₁} using isT1_def by auto
qed

theorem (in topgroup) T0_imp_neu_closed:
  assumes T{is T₀}
  shows {0}{is closed in}T
proof-
  {
    fix x assume x∈cl({0}) and x≠0
    have cl({0})=(⋂H∈𝒩₀. 0+H) using cl_point zero_in_tgroup by auto
    moreover
    {
      fix H  assume H∈𝒩₀
      then have H⊆G unfolding zerohoods_def by auto
      then have 0+H=H using image_id_same  group0.trans_neutral(2) group0_valid_in_tgroup
unfolding gzero_def ltrans_def
        by auto
      with ‹H∈𝒩₀› have 0+H∈𝒩₀ H∈{0+H. H∈𝒩₀} by auto
    }
    then have {0+H. H∈𝒩₀}=𝒩₀ by blast
    ultimately have cl({0})=(⋂𝒩₀) by auto
    from ‹x≠0› and ‹x∈cl({0})› obtain U where U∈T and (x∉U∧0∈U)∨(0∉U∧x∈U)
using assms Top_3_L11(1)
      zero_in_tgroup unfolding isT0_def G_def by blast moreover
    {
      assume 0∈U
      with ‹U∈T› have U∈𝒩₀ using zerohoods_def G_def Top_2_L3 by auto
      with ‹x∈cl({0})› and ‹cl({0})=(⋂𝒩₀)› have x∈U by auto
    }
    ultimately have 0∉U and x∈U by auto
```

```
    with ‹U∈T› ‹x∈cl({0})› have False using cl_inter_neigh zero_in_tgroup
unfolding G_def by blast
  }
  then have cl({0})⊆{0} by auto
  then have cl({0})={0} using zero_in_tgroup cl_contains_set G_def by
blast
  then show thesis using Top_3_L8 zero_in_tgroup unfolding G_def by auto
qed
```

98.2 Existence of nice neighbourhoods.

```
lemma (in topgroup) exist_basehoods_closed:
  assumes U∈𝒩₀
  shows ∃V∈𝒩₀. cl(V)⊆U
proof-
  from assms obtain V where V∈𝒩₀ V⊆U (V+V)⊆U (-V)=V using exists_procls_zerohood
by blast
  have inv_fun:GroupInv(G,f)∈G→G using group0_2_T2 Ggroup by auto
  have f_fun:f∈G×G→G using group0.group_oper_fun group0_valid_in_tgroup
by auto
  {
    fix x assume x∈cl(V)
    with ‹V∈𝒩₀› have x∈⋃T V⊆⋃T using Top_3_L11(1) unfolding zerohoods_def
G_def by blast+
    with ‹V∈𝒩₀› have x∈int(x+V) using elem_in_int_ltrans G_def by auto
    with ‹V⊆⋃T›‹x∈cl(V)› have int(x+V)∩V≠0 using cl_inter_neigh Top_2_L2
by blast
    then have (x+V)∩V≠0 using Top_2_L1 by blast
    then obtain q where q∈(x+V) and q∈V by blast
    with ‹V⊆⋃T›‹x∈⋃T› obtain v where q=x+v v∈V unfolding ltrans_def
grop_def using group0.ltrans_image
      group0_valid_in_tgroup unfolding G_def by auto
    from ‹V⊆⋃T› ‹v∈V›‹q∈V› have v∈⋃T q∈⋃T by auto
    with ‹q=x+v›‹x∈⋃T› have q-v=x using group0.group0_2_L18(1) group0_valid_in_tgroup
unfolding G_def
        unfolding grsub_def grinv_def grop_def by auto moreover
    from ‹v∈V› have (-v)∈(-V) unfolding setninv_def grinv_def using func_imagedef
inv_fun ‹V⊆⋃T› G_def by auto
    then have (-v)∈V using ‹(-V)=V› by auto
    with ‹q∈V› have ⟨q,-v⟩∈V×V by auto
    then have f⟨q,-v⟩∈V+V using lift_subset_suff f_fun ‹V⊆⋃T› unfold-
ing setadd_def by auto
    with ‹V+V⊆U› have q-v∈U unfolding grsub_def grop_def by auto
    with ‹q-v=x› have x∈U by auto
  }
  then have cl(V)⊆U by auto
  with ‹V∈𝒩₀› show thesis by auto
qed
```

98.3 Rest of separation axioms

```
theorem(in topgroup) T1_imp_T2:
  assumes T{is T₁}
  shows T{is T₂}
proof-
  {
    fix x y assume ass:x∈⋃T y∈⋃T x≠y
    {
      assume (-y)+x=0
      with ass(1,2) have y=x using group0.group0_2_L11[where a=y and
b=x] group0_valid_in_tgroup by auto
      with ass(3) have False by auto
    }
    then have (-y)+x≠0 by auto
    then have 0≠(-y)+x by auto
    from ‹y∈⋃T› have (-y)∈⋃T using neg_in_tgroup G_def by auto
    with ‹x∈⋃T› have (-y)+x∈⋃T using group0.group_op_closed[where
a=-y and b=x] group0_valid_in_tgroup unfolding
      G_def by auto
    with assms ‹0≠(-y)+x› obtain U where U∈T and (-y)+x∉U and 0∈U un-
folding isT1_def using zero_in_tgroup
      by auto
    then have U∈𝒩₀ unfolding zerohoods_def G_def using Top_2_L3 by auto
    then obtain Q where Q∈𝒩₀ Q⊆U (Q+Q)⊆U (-Q)=Q using exists_procls_zerohood
by blast
    with ‹(-y)+x∉U› have (-y)+x∉Q by auto
    from ‹Q∈𝒩₀› have Q⊆G unfolding zerohoods_def by auto
    {
      assume x∈y+Q
      with ‹Q⊆G› ‹y∈⋃T› obtain u where u∈Q and x=y+u unfolding ltrans_def
grop_def using group0.ltrans_image group0_valid_in_tgroup
        unfolding G_def by auto
      with ‹Q⊆G› have u∈⋃T unfolding G_def by auto
      with ‹x=y+u› ‹y∈⋃T› ‹x∈⋃T› ‹Q⊆G› have (-y)+x=u using group0.group0_2_L18(2)
group0_valid_in_tgroup unfolding G_def
        unfolding grsub_def grinv_def grop_def by auto
      with ‹u∈Q› have (-y)+x∈Q by auto
      then have False using ‹(-y)+x∉Q› by auto
    }
    then have x∉y+Q by auto moreover
    {
      assume y∈x+Q
      with ‹Q⊆G› ‹x∈⋃T› obtain u where u∈Q and y=x+u unfolding ltrans_def
grop_def using group0.ltrans_image group0_valid_in_tgroup
        unfolding G_def by auto
      with ‹Q⊆G› have u∈⋃T unfolding G_def by auto
      with ‹y=x+u› ‹y∈⋃T› ‹x∈⋃T› ‹Q⊆G› have (-x)+y=u using group0.group0_2_L18(2)
group0_valid_in_tgroup unfolding G_def
        unfolding grsub_def grinv_def grop_def by auto
```

```
      with ‹u∈Q› have (-y)+x=-u using group0.group_inv_of_two[OF group0_valid_in_tgroup
group0.inverse_in_group[OF group0_valid_in_tgroup,of x],of y]
        using ‹x∈⋃T› ‹y∈⋃T› using group0.group_inv_of_inv[OF group0_valid_in_tgroup]
unfolding G_def grinv_def grop_def by auto
      moreover from ‹u∈Q› have (-u)∈(-Q) unfolding setninv_def grinv_def
using func_imagedef[OF group0_2_T2[OF Ggroup] ‹Q⊆G›] by auto
      ultimately have (-y)+x∈Q using ‹(-y)+x∉Q› ‹(-Q)=Q› unfolding setninv_def
grinv_def by auto
      then have False using ‹(-y)+x∉Q› by auto
    }
    then have y∉x+Q by auto moreover
    {
      fix t
      assume t∈(x+Q)∩(y+Q)
      then have t∈(x+Q) t∈(y+Q) by auto
      with ‹Q⊆G› ‹x∈⋃T› ‹y∈⋃T› obtain u v where u∈Q v∈Q and t=x+u
t=y+v unfolding ltrans_def grop_def using group0.ltrans_image[OF group0_valid_in_tgroup]
        unfolding G_def by auto
      then have x+u=y+v by auto
      moreover from ‹u∈Q› ‹v∈Q› ‹Q⊆G› have u∈⋃T v∈⋃T unfolding G_def
by auto
      moreover note ‹x∈⋃T› ‹y∈⋃T›
      ultimately have (-y)+(x+u)=v using group0.group0_2_L18(2)[OF group0_valid_in_tgroup,
of y v x+u] group0.group_op_closed[OF group0_valid_in_tgroup, of x u]
unfolding G_def
        unfolding grsub_def grinv_def grop_def by auto
      then have ((-y)+x)+u=v using group0.group_oper_assoc[OF group0_valid_in_tgroup]
        unfolding grop_def using ‹x∈⋃T› ‹y∈⋃T› ‹u∈⋃T› using group0.inverse_in_group[OF
group0_valid_in_tgroup] unfolding G_def
        by auto
      then have ((-y)+x)=v-u using group0.group0_2_L18(1)[OF group0_valid_in_tgroup,of
(-y)+x u v]
        using ‹(-y)+x∈⋃T› ‹u∈⋃T› ‹v∈⋃T› unfolding G_def grsub_def grinv_def
grop_def by force
      moreover
      from ‹u∈Q› have (-u)∈(-Q) unfolding setninv_def grinv_def using
func_imagedef[OF group0_2_T2[OF Ggroup] ‹Q⊆G›] by auto
      then have (-u)∈Q using ‹(-Q)=Q› by auto
      with ‹v∈Q› have ⟨v,-u⟩∈Q×Q by auto
      then have f⟨v,-u⟩∈Q+Q using lift_subset_suff[OF group0.group_oper_fun[OF
group0_valid_in_tgroup] ‹Q⊆G› ‹Q⊆G›]
        unfolding setadd_def by auto
      with ‹Q+Q⊆U› have v-u∈U unfolding grsub_def grop_def by auto
      ultimately have (-y)+x∈U by auto
      with ‹(-y)+x∉U› have False by auto
    }
    then have (x+Q)∩(y+Q)=0 by auto
    moreover have x∈int(x+Q)y∈int(y+Q) using elem_in_int_ltrans ‹Q∈𝒩₀›
      ‹x∈⋃T› ‹y∈⋃T› unfolding G_def by auto moreover
```

```
    have int(x+Q)⊆(x+Q)int(y+Q)⊆(y+Q) using Top_2_L1 by auto
    moreover have int(x+Q)∈T int(y+Q)∈T using Top_2_L2 by auto
    ultimately have int(x+Q)∈T ∧ int(y+Q)∈T ∧ x ∈ int(x+Q) ∧ y ∈ int(y+Q)
∧ int(x+Q) ∩ int(y+Q) = 0
      by blast
    then have ∃U∈T. ∃V∈T. x∈U∧y∈V∧U∩V=0 by auto
  }
  then show thesis using isT2_def by auto
qed
```

Here follow some auxiliary lemmas.

```
lemma (in topgroup) trans_closure:
  assumes x∈G A⊆G
  shows cl(x+A)=x+cl(A)
proof-
  have ⋃T-(⋃T-(x+A))=(x+A) unfolding ltrans_def using group0.group0_5_L1(2)[OF
group0_valid_in_tgroup assms(1)]
    unfolding image_def range_def domain_def converse_def Pi_def by auto
  then have cl(x+A)=⋃T-int(⋃T-(x+A)) using Top_3_L11(2)[of ⋃T-(x+A)]
by auto moreover
  have x+G=G using surj_image_eq group0.trans_bij(2)[OF group0_valid_in_tgroup
assms(1)] bij_def by auto
  then have ⋃T-(x+A)=x+(⋃T-A) using inj_image_dif[of LeftTranslation(G,
f, x)GG, OF _ assms(2)]
    unfolding ltrans_def G_def using group0.trans_bij(2)[OF group0_valid_in_tgroup
assms(1)] bij_def by auto
  then have int(⋃T-(x+A))=int(x+(⋃T-A)) by auto
  then have int(⋃T-(x+A))=x+int(⋃T-A) using ltrans_interior[OF assms(1),of
⋃T-A] unfolding G_def by force
  have ⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A)) using Top_3_L11(2)[of ⋃T-A] by
force
  have ⋃T-(⋃T-A)=A using assms(2) G_def by auto
  with ‹⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A))› have ⋃T-int(⋃T-A)=cl(A) by auto
  have ⋃T-(⋃T-int(⋃T-A))=int(⋃T-A) using Top_2_L2 by auto
  with ‹⋃T-int(⋃T-A)=cl(A)› have int(⋃T-A)=⋃T-cl(A) by auto
  with ‹int(⋃T-(x+A))=x+int(⋃T-A)› have int(⋃T-(x+A))=x+(⋃T-cl(A))
by auto
  with ‹x+G=G› have int(⋃T-(x+A))=⋃T-(x+cl(A)) using inj_image_dif[of
LeftTranslation(G, f, x)GGcl(A)]
    unfolding ltrans_def using group0.trans_bij(2)[OF group0_valid_in_tgroup
assms(1)] Top_3_L11(1) assms(2) unfolding bij_def G_def
    by auto
  then have ⋃T-int(⋃T-(x+A))=⋃T-(⋃T-(x+cl(A))) by auto
  then have ⋃T-int(⋃T-(x+A))=x+cl(A) unfolding ltrans_def using group0.group0_5_L1(2)[OF
group0_valid_in_tgroup assms(1)]
    unfolding image_def range_def domain_def converse_def Pi_def by auto
  with ‹cl(x+A)=⋃T-int(⋃T-(x+A))› show thesis by auto
qed
```

```
lemma (in topgroup) trans_interior2: assumes A1: g∈G and A2: A⊆G
  shows int(A)+g = int(A+g)
proof -
  from assms have A ⊆ ⋃T and IsAhomeomorphism(T,T,RightTranslation(G,f,g))
    using tr_homeo by auto
  then show thesis using int_top_invariant by simp
qed

lemma (in topgroup) trans_closure2:
  assumes x∈G A⊆G
  shows cl(A+x)=cl(A)+x
proof-
  have ⋃T-(⋃T-(A+x))=(A+x) unfolding ltrans_def using group0.group0_5_L1(1)[OF
group0_valid_in_tgroup assms(1)]
    unfolding image_def range_def domain_def converse_def Pi_def by auto
  then have cl(A+x)=⋃T-int(⋃T-(A+x)) using Top_3_L11(2)[of ⋃T-(A+x)]
by auto moreover
  have G+x=G using surj_image_eq group0.trans_bij(1)[OF group0_valid_in_tgroup
assms(1)] bij_def by auto
  then have ⋃T-(A+x)=(⋃T-A)+x using inj_image_dif[of RightTranslation(G,
f, x)GG, OF _ assms(2)]
    unfolding rtrans_def G_def using group0.trans_bij(1)[OF group0_valid_in_tgroup
assms(1)] bij_def by auto
  then have int(⋃T-(A+x))=int((⋃T-A)+x) by auto
  then have int(⋃T-(A+x))=int(⋃T-A)+x using trans_interior2[OF assms(1),of
⋃T-A] unfolding G_def by force
  have ⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A)) using Top_3_L11(2)[of ⋃T-A] by
force
  have ⋃T-(⋃T-A)=A using assms(2) G_def by auto
  with ‹⋃T-int(⋃T-A)=cl(⋃T-(⋃T-A))› have ⋃T-int(⋃T-A)=cl(A) by auto
  have ⋃T-(⋃T-int(⋃T-A))=int(⋃T-A) using Top_2_L2 by auto
  with ‹⋃T-int(⋃T-A)=cl(A)› have int(⋃T-A)=⋃T-cl(A) by auto
  with ‹int(⋃T-(A+x))=int(⋃T-A)+x› have int(⋃T-(A+x))=(⋃T-cl(A))+x
by auto
  with ‹G+x=G› have int(⋃T-(A+x))=⋃T-(cl(A)+x) using inj_image_dif[of
RightTranslation(G, f, x)GGcl(A)]
    unfolding rtrans_def using group0.trans_bij(1)[OF group0_valid_in_tgroup
assms(1)] Top_3_L11(1) assms(2) unfolding bij_def G_def
    by auto
  then have ⋃T-int(⋃T-(A+x))=⋃T-(⋃T-(cl(A)+x)) by auto
  then have ⋃T-int(⋃T-(A+x))=cl(A)+x unfolding ltrans_def using group0.group0_5_L1(1)[OF
group0_valid_in_tgroup assms(1)]
    unfolding image_def range_def domain_def converse_def Pi_def by auto
  with ‹cl(A+x)=⋃T-int(⋃T-(A+x))› show thesis by auto
qed

lemma (in topgroup) trans_subset:
  assumes A⊆((-x)+B)x∈GA⊆GB⊆G
  shows x+A⊆B
```

```
proof-
  {
   fix t assume t∈x+A
    with ‹x∈G› ‹A⊆G› obtain u where u∈A t=x+u unfolding ltrans_def grop_def
using group0.ltrans_image[OF group0_valid_in_tgroup]
      unfolding G_def by auto
    with ‹x∈G› ‹A⊆G› ‹u∈A› have (-x)+t=u using group0.group0_2_L18(2)[OF
group0_valid_in_tgroup, of xut]
      group0.group_op_closed[OF group0_valid_in_tgroup,of x u] unfold-
ing grop_def grinv_def by auto
    with ‹u∈A› have (-x)+t∈A by auto
    with ‹A⊆(-x)+B› have (-x)+t∈(-x)+B by auto
    with ‹B⊆G› obtain v where (-x)+t=(-x)+v v∈B unfolding ltrans_def
grop_def using neg_in_tgroup[OF ‹x∈G›] group0.ltrans_image[OF group0_valid_in_tgroup]
      unfolding G_def by auto
    have LeftTranslation(G,f,-x)∈inj(G,G) using group0.trans_bij(2)[OF
group0_valid_in_tgroup neg_in_tgroup[OF ‹x∈G›]] bij_def by auto
    then have eq:∀A∈G. ∀B∈G. LeftTranslation(G,f,-x)A=LeftTranslation(G,f,-
x)B ⟶ A=B unfolding inj_def by auto
    {
      fix A B assume A∈GB∈G
      assume f⟨-x,A⟩=f⟨-x,B⟩
      then have LeftTranslation(G,f,-x)A=LeftTranslation(G,f,-x)B us-
ing group0.group0_5_L2(2)[OF group0_valid_in_tgroup neg_in_tgroup[OF ‹x∈G›]]
        ‹A∈G›‹B∈G› by auto
      with eq ‹A∈G›‹B∈G› have A=B by auto
    }
    then have eq1:∀A∈G. ∀B∈G. f⟨-x,A⟩=f⟨-x,B⟩ ⟶ A=B by auto
    from ‹A⊆G› ‹u∈A› have u∈G by auto
    with ‹v∈B› ‹B⊆G› ‹t=x+u› have t∈G v∈G using group0.group_op_closed[OF
group0_valid_in_tgroup ‹x∈G›,of u] unfolding grop_def
      by auto
    with eq1 ‹(-x)+t=(-x)+v› have t=v unfolding grop_def by auto
    with ‹v∈B› have t∈B by auto
  }
  then show thesis by auto
qed
```

Every topological group is regular, and hence T_3. The proof is in the next section, since it uses local properties.

98.4 Local properties

In a topological group, all local properties depend only on the neighbourhoods of the neutral element; when considering topological properties. The next result of regularity, will use this idea, since translations preserve closed sets.

```
lemma (in topgroup) local_iff_neutral:
```

```
  assumes ∀U∈T∩𝒩0. ∃N∈𝒩0. N⊆U∧ P(N,T) ∀N∈Pow(G). ∀x∈G. P(N,T) ⟶
P(x+N,T)
  shows T{is locally}P
proof-
  {
    fix x U assume x∈⋃TU∈Tx∈U
    then have (-x)+U∈T∩𝒩0 using open_tr_open(1) open_trans_neigh neg_in_tgroup
unfolding G_def
      by auto
    with assms(1) obtain N where N⊆((-x)+U)P(N,T)N∈𝒩0 by auto
    note ‹x∈⋃T›‹N⊆((-x)+U)› moreover
    from ‹U∈T› have U⊆⋃T by auto moreover
    from ‹N∈𝒩0› have N⊆G unfolding zerohoods_def by auto
    ultimately have (x+N)⊆U using trans_subset unfolding G_def by auto
moreover
    from ‹N⊆G›‹x∈⋃T› assms(2) ‹P(N,T)› have P((x+N),T) unfolding G_def
by auto moreover
    from ‹N∈𝒩0›‹x∈⋃T› have x∈int(x+N) using elem_in_int_ltrans un-
folding G_def by auto
    ultimately have ∃N∈Pow(U). x∈int(N)∧P(N,T) by auto
  }
  then show thesis unfolding IsLocally_def[OF topSpaceAssum] by auto
qed

lemma (in topgroup) trans_closed:
  assumes A{is closed in}Tx∈G
  shows (x+A){is closed in}T
proof-
  from assms(1) have cl(A)=A using Top_3_L8 unfolding IsClosed_def by
auto
  then have x+cl(A)=x+A by auto
  then have cl(x+A)=x+A using trans_closure assms unfolding IsClosed_def
by auto
  moreover have x+A⊆G unfolding ltrans_def using group0.group0_5_L1(2)[OF
group0_valid_in_tgroup ‹x∈G›]
      unfolding image_def range_def domain_def converse_def Pi_def by
auto
  ultimately show thesis using Top_3_L8 unfolding G_def by auto
qed
```

As it is written in the previous section, every topological group is regular.

```
theorem (in topgroup) topgroup_reg:
  shows T{is regular}
proof-
  {
    fix U assume U∈T∩𝒩0
    then obtain V where cl(V)⊆UV∈𝒩0 using exist_basehoods_closed by
blast
    then have V⊆cl(V) using cl_contains_set unfolding zerohoods_def G_def
```

```
by auto
    then have int(V)⊆int(cl(V)) using interior_mono by auto
    with ‹V∈𝒩₀› have cl(V)∈𝒩₀ unfolding zerohoods_def G_def using Top_3_L11(1)
by auto
    from ‹V∈𝒩₀› have cl(V){is closed in}T using cl_is_closed unfold-
ing zerohoods_def G_def by auto
    with ‹cl(V)∈𝒩₀›‹cl(V)⊆U› have ∃N∈𝒩₀. N⊆U∧N{is closed in}T by auto
  }
  then have ∀U∈T∩𝒩₀. ∃N∈𝒩₀. N⊆U∧N{is closed in}T by auto moreover
  have ∀N∈Pow(G).( ∀x∈G. (N{is closed in}T⟶(x+N){is closed in}T))
using trans_closed by auto
  ultimately have T{is locally-closed} using local_iff_neutral unfold-
ing IsLocallyClosed_def by auto
  then show T{is regular} using regular_locally_closed by auto
qed
```

The promised corollary follows:

```
corollary (in topgroup) T2_imp_T3:
  assumes T{is T₂}
  shows T{is T₃} using T2_is_T1 T1_is_T0 topgroup_reg isT3_def assms by
auto

end
```

99 Topological groups - uniformity

```
theory TopologicalGroup_Uniformity_ZF imports TopologicalGroup_ZF UniformSpace_ZF_1

begin
```

Each topological group is a uniform space. This theory is about the unifomities that are naturally defined by a topological group structure.

99.1 Natural uniformities in topological groups: definitions and notation

There are two basic uniformities that can be defined on a topological group.

Definition of left uniformity

```
definition (in topgroup) leftUniformity
 where leftUniformity ≡ {V∈Pow(G×G).∃U∈ 𝒩₀. {⟨s,t⟩∈G×G. (-s)+t ∈U}
⊆ V}
```

Definition of right uniformity

```
definition (in topgroup) rightUniformity
 where rightUniformity ≡ {V∈Pow(G×G).∃U∈ 𝒩₀. {⟨s,t⟩∈G×G. s+(-t) ∈U}⊆
V}
```

Right and left uniformities are indeed uniformities.

```
lemma (in topgroup) side_uniformities:
    shows leftUniformity {is a uniformity on} G and rightUniformity {is
a uniformity on} G
proof-
  {
    assume 0 ∈ leftUniformity
    then obtain U where U:U∈𝒩₀ {⟨s,t⟩∈G×G. (-s)+t ∈U}⊆0 unfolding leftUniformity_def

      by auto
    have ⟨0,0⟩:G×G using zero_in_tgroup by auto
    moreover have (-0)+0 = 0
      using group0_valid_in_tgroup group0.group_inv_of_one group0.group0_2_L2
zero_in_tgroup
      by auto
    moreover have 0∈int(U) using U(1) by auto
    then have 0∈U using Top_2_L1 by auto
    ultimately have ⟨0,0⟩∈{⟨s,t⟩∈G×G. (-s)+t ∈U} by auto
    with U(2) have ⟨0,0⟩∈0 by blast
    hence False by auto
  }
  hence 0∉leftUniformity by auto
  moreover  have leftUniformity ⊆ Pow(G×G) unfolding leftUniformity_def
by auto
  moreover
  {
    have G×G∈Pow(G×G) by auto moreover
    have {⟨s,t⟩:G×G. (-s)+t:G} ⊆ G×G by auto moreover
    note zneigh_not_empty
    ultimately have G×G∈leftUniformity unfolding leftUniformity_def by
auto
  }
  moreover
  {
    fix A B assume as:A∈leftUniformity B∈leftUniformity
    from as(1) obtain AU where AU:AU∈ 𝒩₀ {⟨s,t⟩∈G×G. (-s)+t ∈AU}⊆ A
A∈Pow(G×G)
      unfolding leftUniformity_def by auto
    from as(2) obtain BU where BU:BU∈ 𝒩₀ {⟨s,t⟩∈G×G. (-s)+t ∈BU}⊆ B
B∈Pow(G×G)
      unfolding leftUniformity_def by auto
    from AU(1) BU(1) have 0∈int(AU)∩int(BU) by auto
    moreover from AU BU have op:int(AU)∩int(BU)∈T using Top_2_L2 topSpaceAssum
IsATopology_def
      by auto
    moreover
    have int(AU)∩int(BU) ⊆ AU∩BU using Top_2_L1 by auto
    with op have int(AU)∩int(BU)⊆int(AU∩BU) using Top_2_L5 by auto
    moreover note AU(1) BU(1)
```

```
    ultimately have AU∩BU: 𝒩₀ unfolding zerohoods_def by auto
    moreover have {⟨s,t⟩∈G×G. (-s)+t ∈AU∩BU}⊆{⟨s,t⟩∈G×G. (-s)+t ∈AU}
by auto
    with AU(2) BU(2) have {⟨s,t⟩∈G×G. (-s)+t ∈AU∩BU}⊆A∩B by auto
    ultimately have A∩B ∈ {V∈Pow(G×G).∃U∈ 𝒩₀. {⟨s,t⟩∈G×G. (-s)+t ∈U}
⊆ V}
      using AU(3) BU(3) by blast
    then have A∩B∈leftUniformity unfolding leftUniformity_def by simp

  }
  hence ∀A∈leftUniformity. ∀B∈leftUniformity. A ∩ B ∈ leftUniformity
by auto
  moreover
  {
    fix B C assume as:B∈leftUniformity C∈Pow(G × G) B ⊆ C
    from as(1) obtain BU where BU:BU∈𝒩₀ {⟨s,t⟩∈G×G. (-s)+t ∈ BU}⊆B

      unfolding leftUniformity_def by blast
    from as(3) BU(2) have {⟨s,t⟩∈G×G. (-s)+t ∈ BU}⊆C by auto
    with as(2) BU(1) have C ∈ {V∈Pow(G×G).∃U∈ 𝒩₀. {⟨s,t⟩∈G×G. (-s)+t
∈U} ⊆ V}
      by auto
    then have C ∈ leftUniformity  unfolding leftUniformity_def by auto
  }
  then have ∀B∈leftUniformity. ∀C∈Pow(G×G). B⊆C ⟶ C∈leftUniformity
by auto
  ultimately have leftFilter:leftUniformity {is a filter on} (G×G) un-
folding IsFilter_def
    by auto
  {
    assume 0∈rightUniformity
    then obtain U where U:U∈𝒩₀ {⟨s,t⟩∈G×G. s+(-t) ∈U}⊆0 unfolding rightUniformity_def

      by auto
    have ⟨0,0⟩:G×G using zero_in_tgroup by auto
    moreover have 0+(-0) = 0
      using group0_valid_in_tgroup group0.group_inv_of_one group0.group0_2_L2
zero_in_tgroup
      by auto
    moreover
    have 0∈int(U) using U(1) by auto
    then have 0∈U using Top_2_L1 by auto
    ultimately have ⟨0,0⟩∈{⟨s,t⟩∈G×G. s+(-t) ∈U} by auto
    with U(2) have ⟨0,0⟩∈0 by blast
    hence False by auto
  }
  then have 0∉rightUniformity by auto
  moreover  have rightUniformity ⊆ Pow(G×G) unfolding rightUniformity_def
by auto
```

```
  moreover
  {
    have G×G∈Pow(G×G) by auto
    moreover have {⟨s,t⟩:G×G. (-s)+t:G} ⊆ G×G by auto
    moreover note zneigh_not_empty
    ultimately have G×G ∈ rightUniformity unfolding rightUniformity_def
by auto
  }
  moreover
  {
    fix A B assume as:A∈rightUniformity B∈rightUniformity
    from as(1) obtain AU where AU:AU∈ 𝒩₀ {⟨s,t⟩∈G×G. s+(-t) ∈AU}⊆ A
A∈Pow(G×G)
      unfolding rightUniformity_def by auto
    from as(2) obtain BU where BU:BU∈ 𝒩₀ {⟨s,t⟩∈G×G. s+(-t) ∈BU}⊆ B
B∈Pow(G×G)
      unfolding rightUniformity_def by auto
    from AU(1) BU(1) have 0∈int(AU)∩int(BU) by auto
    moreover from AU BU have op:int(AU)∩int(BU)∈T
      using Top_2_L2 topSpaceAssum IsATopology_def by auto
    moreover
    have int(AU)∩int(BU) ⊆ AU∩BU using Top_2_L1 by auto
    with op have int(AU)∩int(BU)⊆int(AU∩BU) using Top_2_L5 by auto
    moreover note AU(1) BU(1)
    ultimately have AU∩BU: 𝒩₀ unfolding zerohoods_def by auto
    moreover have {⟨s,t⟩∈G×G. s+(-t) ∈AU∩BU}⊆{⟨s,t⟩∈G×G. s+(-t) ∈AU}
by auto
    with AU(2) BU(2) have {⟨s,t⟩∈G×G. s+(-t) ∈AU∩BU}⊆A∩B by auto
    ultimately have A∩B ∈ {V∈Pow(G×G).∃U∈ 𝒩₀. {⟨s,t⟩∈G×G. s+(-t) ∈U}⊆
V}
      using AU(3) BU(3) by blast
    then have A∩B ∈ rightUniformity unfolding rightUniformity_def by
simp
  }
  hence ∀A∈rightUniformity. ∀B∈rightUniformity. A∩B ∈ rightUniformity
by auto
  moreover
  {
    fix B C assume as:B∈rightUniformity C∈Pow(G × G) B ⊆ C
    from as(1) obtain BU where BU:BU∈𝒩₀ {⟨s,t⟩∈G×G. s+(-t) ∈ BU}⊆B

      unfolding rightUniformity_def by blast
    from as(3) BU(2) have {⟨s,t⟩∈G×G. s+(-t) ∈ BU}⊆C by auto
    then have C ∈ rightUniformity using as(2) BU(1) unfolding rightUniformity_def
by auto
  }
  then have ∀B∈rightUniformity. ∀C∈Pow(G×G). B⊆C ⟶ C∈rightUniformity
by auto
  ultimately have rightFilter:rightUniformity {is a filter on} (G×G)
```

```
unfolding IsFilter_def
    by auto
  {
    fix U assume as:U∈leftUniformity
    from as obtain V where V:V∈𝒩0 {⟨s,t⟩∈G×G. (-s)+t ∈ V} ⊆ U
      unfolding leftUniformity_def by auto
    then have V⊆G by auto
    {
      fix x assume as2:x∈id(G)
      from as obtain V where V:V∈𝒩0 {⟨s,t⟩∈G×G. (-s)+t ∈ V} ⊆ U
        unfolding leftUniformity_def by auto
      from V(1) have 0∈int(V) by auto
      then have V0:0∈V using Top_2_L1 by auto
      from as2 obtain t where t:x=⟨t,t⟩ t:G by auto
      from t(2) have (-t)+t =0 using group0_valid_in_tgroup group0.group0_2_L6
        by auto
      with V0 t V(2) have x∈U by auto
    }
    then have id(G)⊆U by auto
    moreover
    {
      {
        fix x assume ass:x∈{⟨s,t⟩∈G×G. (-s)+t ∈ -V}
        then obtain s t where as:s∈G t∈G (-s)+t ∈ -V x=⟨s,t⟩ by force
        from as(3) ‹V⊆G› have (-s)+t∈{-q. q∈V} using ginv_image_add
by simp
        then obtain q where q: q∈V (-s)+t = -q by auto
        with ‹V⊆G› have q∈G by auto
        with ‹s∈G› ‹t∈G› ‹(-s)+t = -q› have q=(-t)+s
          using simple_equation1_add by blast
        with q(1) have (-t)+s ∈ V by auto
        with as(1,2) have ⟨t,s⟩ ∈ U using V(2) by auto
        then have ⟨s,t⟩ ∈ converse(U) by auto
        with as(4) have x ∈ converse(U) by auto
      }
      then have {⟨s,t⟩∈G×G. (-s)+t ∈ -V} ⊆ converse(U) by auto
      moreover have (-V):𝒩0 using neg_neigh_neigh V(1) by auto
      moreover note as
      ultimately have converse(U) ∈ leftUniformity unfolding leftUniformity_def
by auto
    }
    moreover
    {
      from V(1) obtain W where W:W:𝒩0 W + W ⊆V using exists_procls_zerohood
by blast
      {
        fix x assume as:x ∈ {⟨s,t⟩∈G×G. (-s)+t ∈ W} O {⟨s,t⟩∈G×G. (-
s)+t ∈ W}
        then obtain x1 x2 x3 where
```

```
          x:x₁∈G x₂∈G x₃∈G (-x₁)+x₂ ∈ W (-x₂)+x₃ ∈ W x=⟨x₁,x₃⟩
          unfolding comp_def by auto
        from W(1) have W+W = f(W×W) using interval_add(2) by auto
        moreover from W(1) have WW:W×W⊆G×G by auto
        moreover
        from x(4,5) have ⟨(-x₁)+x₂,(-x₂)+x₃⟩:W×W by auto
        with WW have f(⟨(-x₁)+x₂,(-x₂)+x₃⟩):f(W×W)
          using func_imagedef topgroup_f_binop by auto
        ultimately have ((-x₁)+x₂)+((-x₂)+x₃) :W+W by auto
        moreover from x(1,2,3) have ((-x₁)+x₂)+((-x₂)+x₃) = (-x₁)+ x₃
          using cancel_middle_add(2) by simp
        ultimately have (-x₁)+ x₃∈W+W by auto
        with W(2) have (-x₁)+ x₃∈V by auto
        with x(1,3,6) have x:{⟨s,t⟩∈G×G. (-s)+t ∈ V} by auto
      }
      then have {⟨s,t⟩∈G×G. (-s)+t ∈ W} O {⟨s,t⟩∈G×G. (-s)+t ∈ W} ⊆
U
        using V(2) by auto moreover
      have {⟨s,t⟩∈G×G. (-s)+t ∈ W}∈leftUniformity
        unfolding leftUniformity_def using W(1) by auto
      ultimately have ∃Z∈leftUniformity. Z O Z⊆U by auto
    }
    ultimately have id(G)⊆U ∧ (∃Z∈leftUniformity. Z O Z⊆U) ∧ converse(U)∈leftUniformity

      by blast
  }
  then have
    ∀U∈leftUniformity. id(G)⊆U ∧ (∃Z∈leftUniformity. Z O Z⊆U) ∧ converse(U)∈leftUniformi

    by auto
  with leftFilter show leftUniformity {is a uniformity on} G unfold-
ing IsUniformity_def by auto
  {
    fix U assume as:U∈rightUniformity
    from as obtain V where V:V∈𝒩₀ {⟨s,t⟩∈G×G. s+(-t) ∈ V} ⊆ U
      unfolding rightUniformity_def by auto
    {
      fix x assume as2:x∈id(G)
      from as obtain V where V:V∈𝒩₀ {⟨s,t⟩∈G×G. s+(-t) ∈ V} ⊆ U
        unfolding rightUniformity_def by auto
      from V(1) have 0∈int(V) by auto
      then have V0:0∈V using Top_2_L1 by auto
      from as2 obtain t where t:x=⟨t,t⟩ t:G by auto
      from t(2) have t+(-t) =0 using group0_valid_in_tgroup group0.group0_2_L6

        by auto
      with V0 t V(2) have x∈U by auto
    }
    then have id(G)⊆U by auto
```

```
    moreover
    {
      {
        fix x assume ass:x∈{⟨s,t⟩∈G×G. s+(-t) ∈ -V}
        then obtain s t where as:s∈G t∈G s+(-t) ∈ -V x=⟨s,t⟩
          by force
        from as(3) V(1) have s+(-t) ∈ {-q. q∈V}
          using ginv_image_add by simp
        then obtain q where q:q∈V s+(-t) = -q by auto
        with ‹V∈𝒩₀› have q∈G by auto
        with as(1,2) q(1,2) have t+(-s) ∈ V using simple_equation0_add

          by blast
        with as(1,2,4) V(2) have x ∈ converse(U) by auto
      }
      then have {⟨s,t⟩∈G×G.  s+(-t) ∈ -V} ⊆ converse(U) by auto
      moreover from V(1) have (-V) ∈ 𝒩₀ using neg_neigh_neigh by auto
      ultimately have converse(U) ∈ rightUniformity using as rightUniformity_def

        by auto
    }
    moreover
    {
      from V(1) obtain W where W:W:𝒩₀ W + W ⊆V using exists_procls_zerohood
by blast
      {
        fix x assume as:x:{⟨s,t⟩∈G×G.  s+(-t) ∈ W} O {⟨s,t⟩∈G×G.  s+(-
t) ∈ W}
        then obtain x₁ x₂ x₃ where
          x:x₁:G x₂∈G x₃∈G x₁+(-x₂) ∈ W x₂+(-x₃) ∈ W x=⟨x₁,x₃⟩
          unfolding comp_def by auto
        from W(1) have W+W = f(W×W) using interval_add(2) by auto
        moreover from W(1) have WW:W×W⊆G×G by auto
        moreover
        from x(4,5) have ⟨x₁+(-x₂),x₂+(-x₃)⟩ ∈ W×W by auto
        with WW have f(⟨x₁+(-x₂),x₂+(-x₃)⟩) ∈ f(W×W)
          using func_imagedef topgroup_f_binop  by auto
        ultimately have (x₁+(-x₂))+(x₂+(-x₃)) ∈ W+W by auto
        moreover from x(1,2,3) have (x₁+(-x₂))+(x₂+(-x₃)) = x₁+ (-x₃)
          using cancel_middle_add(1) by simp
        ultimately have x₁+(-x₃) ∈ W+W by auto
        with W(2) have x₁+(-x₃) ∈ V by auto
        then have x ∈ {⟨s,t⟩∈G×G. s+(-t) ∈ V} using x(1,3,6) by auto
      }
      with V(2) have {⟨s,t⟩∈G×G. s+(-t) ∈ W} O {⟨s,t⟩∈G×G. s+(-t) ∈
W} ⊆ U
        by auto
      moreover from W(1) have {⟨s,t⟩∈G×G. s+(-t) ∈ W} ∈ rightUniformity
```

```
      unfolding rightUniformity_def  by auto
    ultimately have ∃Z∈rightUniformity. Z O Z⊆U by auto
  }
  ultimately have id(G)⊆U ∧ (∃Z∈rightUniformity. Z O Z⊆U) ∧ converse(U)∈rightUniformity

    by blast
}
then have
  ∀U∈rightUniformity. id(G)⊆U ∧ (∃Z∈rightUniformity. Z O Z⊆U) ∧ converse(U)∈rightUnifor

  by auto
with rightFilter show rightUniformity {is a uniformity on} G unfold-
ing IsUniformity_def
  by auto
qed
```

The topologies generated by the right and left uniformities are the original group topology.

```
lemma (in topgroup) top_generated_side_uniformities:
  shows UniformTopology(leftUniformity,G) = T and UniformTopology(rightUniformity,G)
= T
proof-
  let M = {⟨t, {V  {t} . V ∈ leftUniformity}⟩. t∈G}
  have fun:M:G→Pow(Pow(G)) using neigh_from_uniformity side_uniformities(1)
IsNeighSystem_def
    by auto
  let N = {⟨t, {V  {t} . V ∈ rightUniformity}⟩. t∈G}
  have funN:N:G→Pow(Pow(G)) using neigh_from_uniformity side_uniformities(2)
IsNeighSystem_def
    by auto
  {
    fix U assume op:U∈T
    hence U⊆G by auto
    {
      fix x assume x:x∈U
      with op have xg:x∈G and (-x) ∈ G using neg_in_tgroup by auto
      then have ⟨x, {V{x}. V ∈ leftUniformity}⟩ ∈ {⟨t, {V{t}. V ∈ leftUniformity}⟩.
t∈G}
        by auto
      with fun have app:M(x) = {V{x}. V ∈ leftUniformity} using ZF_fun_from_tot_val

        by auto
      have (-x)+U : 𝒩₀ using open_trans_neigh op x by auto
      then have V:{⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)} ∈ leftUniformity
        unfolding leftUniformity_def by auto
      with xg have
        N:∀t∈G. t:{⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)}{x} ⟷ (-x)+t∈((-x)+U)
        using image_iff by auto
      {
```

```
        fix t assume t:t∈G
        {
          assume as:(-x)+t∈((-x)+U)
          then have (-x)+t∈LeftTranslation(G,f,-x)U by auto
          then obtain q where q:q∈U ⟨q,(-x)+t⟩∈LeftTranslation(G,f,-
x)
            using image_iff by auto
          with op have q∈G by auto
          from q(2) have (-x)+q = (-x)+t unfolding LeftTranslation_def

            by auto
          with ‹(-x) ∈ G› ‹q∈G› ‹t∈G› have q = t using neg_in_tgroup
cancel_left_add
            by blast
          with q(1) have t∈U by auto
        }
        moreover
        {
          assume t:t∈U
          with ‹U⊆G› ‹(-x)∈G› have (-x)+t ∈ ((-x)+U)
            using lrtrans_image(1) by auto
        }
        ultimately have (-x)+t∈((-x)+U) ⟷ t:U by blast
      }
      with N have ∀t∈G. t:{⟨s,t⟩∈G×G. (-s)+t ∈ ((-x)+U)}{x} ⟷ t∈U

        by blast
      with op have ∀t. t:{⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)}{x} ⟷ t:U
        by auto
      hence U = {⟨s,t⟩∈G×G. (-s)+t∈((-x)+U)}{x} by auto
      with V have ∃V∈leftUniformity. U=V{x} by blast
      with app have U ∈ {⟨t, {V  {t} . V ∈ leftUniformity}⟩ . t ∈ G}(x)
by auto
      moreover from ‹x∈G› funN have app:N(x) = {V{x}. V ∈ rightUniformity}

        using ZF_fun_from_tot_val by simp
      moreover
      from x op have openTrans:U+(-x): 𝒩₀ using open_trans_neigh_2 by
auto
      then have V:{⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))} ∈ rightUniformity
        unfolding rightUniformity_def by auto
      with xg have
        N:∀t∈G. t:{⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))}-{x} ⟷ t+(-x)∈(U+(-
x))
        using vimage_iff by auto
      moreover
      {
        fix t assume t:t∈G
        {
```

```
          assume as:t+(-x)∈(U+(-x))
          hence t+(-x)∈RightTranslation(G,f,-x)U by auto
          then obtain q where q:q∈U ⟨q,t+(-x)⟩∈RightTranslation(G,f,-
x)
            using image_iff by auto
          with op have q∈G by auto
          from q(2) have q+(-x) = t+(-x) unfolding RightTranslation_def
by auto
          with ‹q∈G› ‹(-x) ∈ G› ‹t∈G› have q = t using cancel_right_add
by simp
          with q(1) have t∈U by auto
        }
        moreover
        {
          assume t∈U
          with ‹(-x)∈G› ‹U⊆G› have t+(-x)∈(U+(-x)) using lrtrans_image(2)
            by auto
        } ultimately have t+(-x)∈(U+(-x)) ⟷ t:U by blast
      } with N have ∀t∈G. t:{⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))}-{x} ⟷ t:U

        by blast
      with op have ∀t. t:{⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))}-{x} ⟷ t:U
        by auto
      hence {⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))}-{x} = U by auto
      then have U = converse({⟨s,t⟩∈G×G. s+(-t)∈(U+(-x))}){x}
        unfolding vimage_def by simp
      with V app have U ∈ {⟨t, {V  {t} . V ∈ rightUniformity}⟩ . t ∈
G}(x)
        using side_uniformities(2) IsUniformity_def by auto
      ultimately have
        U ∈ {⟨t, {V  {t} . V ∈ leftUniformity}⟩ . t ∈ G}(x) and
        U ∈ {⟨t, {V  {t} . V ∈ rightUniformity}⟩ . t ∈ G}(x)
        by auto
    }
    hence
      ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ leftUniformity}⟩ . t ∈ G}  x and
      ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ rightUniformity}⟩ . t ∈ G}  x
      by auto
  }
  hence
    T⊆{U ∈ Pow(G) . ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ leftUniformity}⟩ . t
∈ G}  x} and
    T⊆{U ∈ Pow(G) . ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ rightUniformity}⟩ .
t ∈ G}  x}
    by auto
  moreover
  {
    fix U assume as:U ∈ Pow(G) ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ leftUniformity}⟩
. t ∈ G}(x)
```

```
    {
      fix x assume x:x∈U
      with as(1) have xg:x∈G by auto
      from x as(2) have U∈{⟨t, {V  {t} . V ∈ leftUniformity}⟩ . t ∈ G}(x)
by auto
      with xg fun have U∈{V  {x} . V ∈ leftUniformity} using apply_equality
by auto
      then obtain V where V:U=V{x} V∈leftUniformity by auto
      from V(2) obtain W where W:W∈ 𝒩0 {⟨s,t⟩:G×G. (-s)+t:W}⊆V
        unfolding leftUniformity_def by auto
      from W(2) have A:{⟨s,t⟩:G×G. (-s)+t:W}{x}⊆V{x} by auto
      from xg have ∀q∈G. q∈({⟨s,t⟩:G×G. (-s)+t:W}{x}) ⟷ (-x)+q:W
        using image_iff by auto
      hence B:{⟨s,t⟩:G×G. (-s)+t:W}{x} = {t∈G. (-x)+t:W} by auto
      from W(1) have WG:W⊆G by auto
      {
        fix t assume t:t ∈ x+W
        then have t ∈ LeftTranslation(G,f,x)W by auto
        then obtain s where s:s∈W ⟨s,t⟩∈LeftTranslation(G,f,x) using
image_iff by auto
        with ‹W⊆G› have s∈G by auto
        from s(2) have t=x+s t∈G unfolding LeftTranslation_def by auto
        with ‹x∈G› ‹s∈G› have (-x)+t = s using put_on_the_other_side(2)
by simp
        with s(1) have (-x)+t∈W by auto
        with ‹t∈G› have t ∈ {s∈G. (-x)+s:W} by auto
      }
      then have x+W ⊆ {t∈G. (-x)+t∈W} by auto
      with B have x + W ⊆ {⟨s,t⟩ ∈ G × G . (- s) + t ∈ W}  {x} by auto
      with A have x + W ⊆ V {x} by blast
      with V(1) have x + W ⊆ U by auto
      then have int(x + W) ⊆ U using Top_2_L1 by blast
      moreover from xg W(1) have x∈int(x + W) using elem_in_int_ltrans(1)
by auto
      moreover have int(x + W)∈T using Top_2_L2 by auto
      ultimately have ∃Y∈T. x∈Y ∧ Y⊆U by auto
    }
    then have U∈T using open_neigh_open by auto
  } hence  {U ∈ Pow(G) . ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ leftUniformity}⟩
. t ∈ G}  x} ⊆ T
    by auto
  moreover
  {
    fix U assume as:U ∈ Pow(G) ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ rightUniformity}⟩
. t ∈ G}  x
    {
      fix x assume x:x∈U
      with as(1) have xg:x∈G by auto
      from x as(2) have U∈{⟨t, {V  {t} . V ∈ rightUniformity}⟩ . t ∈
```

```
G}  x by auto
      with xg funN have U∈{V  {x} . V ∈ rightUniformity} using apply_equality

        by auto
      then obtain V where V:U=V{x} V ∈ rightUniformity by auto
      then have converse(V) ∈ rightUniformity using side_uniformities(2)
IsUniformity_def
        by auto
      then obtain W where W:W∈ 𝒩₀ {⟨s,t⟩:G×G. s+(-t):W}⊆converse(V)

        unfolding rightUniformity_def by auto
      from W(2) have A:{⟨s,t⟩:G×G. s+(-t):W}-{x}⊆V{x} by auto
      from xg have ∀q∈G. q∈({⟨s,t⟩:G×G. s+(-t):W}-{x}) ⟷ q+(-x):W
        using image_iff by auto
      hence B:{⟨s,t⟩:G×G. s+(-t):W}-{x} = {t∈G. t+(-x):W} by auto
      from W(1) have WG:W⊆G by auto
      {
        fix t assume t ∈ W+x
        with ‹x∈G› ‹W⊆G› obtain s where s∈W and t=s+x using lrtrans_image(2)

          by auto
        with ‹W⊆G› have s∈G by auto
        with ‹x∈G› ‹t=s+x› have t∈G using group_op_closed_add by simp

        from ‹x∈G› ‹s∈G› ‹t=s+x› have t+(-x) = s using put_on_the_other_side

          by simp
        with ‹s∈W› ‹t∈G› have t ∈ {s∈G. s+(-x) ∈ W} by auto
      }
      then have W+x ⊆ {t:G. t+(-x):W} by auto
      with B have W + x ⊆ {⟨s,t⟩ ∈ G × G . s + (- t) ∈ W}-{x} by auto
      with A have W + x ⊆ V {x} by blast
      with V(1) have W + x ⊆ U by auto
      then have int(W + x) ⊆ U using Top_2_L1 by blast
      moreover
      from xg W(1) have x∈int(W + x) using elem_in_int_rtrans(1) by auto
      moreover have int(W + x)∈T using Top_2_L2 by auto
      ultimately have ∃Y∈T. x∈Y ∧ Y⊆U by auto
    }
    then have U∈T using open_neigh_open by auto
  }
  ultimately have
  {U ∈ Pow(G). ∀x∈U. U ∈ {⟨t, {V{t} . V ∈ leftUniformity}⟩. t ∈ G}(x)}
= T
  {U ∈ Pow(G). ∀x∈U. U ∈ {⟨t, {V{t} . V ∈ rightUniformity}⟩. t ∈ G}(x)}
= T
    by auto
  then show UniformTopology(leftUniformity,G) = T and UniformTopology(rightUniformity,G)
= T
```

```
    using uniftop_def_alt by auto
qed
```

The side uniformities are called this way because of how they affect left and right translations. In the next lemma we show that left translations are uniformly continuous with respect to the left uniformity.

```
lemma (in topgroup) left_mult_uniformity: assumes x∈G
  shows
    LeftTranslation(G,f,x) {is uniformly continuous between} leftUniformity
{and} leftUniformity
proof -
  let P = ProdFunction(LeftTranslation(G, f, x), LeftTranslation(G, f,
x))
  from assms have L: LeftTranslation(G,f,x):G→G and leftUniformity {is
a uniformity on} G
    using group0_5_L1 side_uniformities(1) by auto
  moreover have ∀V ∈ leftUniformity. P-(V) ∈ leftUniformity
  proof -
    { fix V assume V ∈ leftUniformity
      then obtain U where U ∈ 𝒩₀ and {⟨s,t⟩ ∈ G × G . (- s) + t ∈ U}
⊆ V
        unfolding leftUniformity_def by auto
      with ‹V ∈ leftUniformity› have
        as:V ⊆ G × G U ∈ 𝒩₀ {⟨s,t⟩ ∈ G × G . (- s) + t ∈ U} ⊆ V
        unfolding leftUniformity_def by auto
      { fix z assume z:z ∈ {⟨s,t⟩ ∈ G × G . (- s) + t ∈ U}
        then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
        from st(1) z have st2: (- s) + t ∈ U by auto
        from assms st have
          P(z) = ⟨LeftTranslation(G, f, x)(s), LeftTranslation(G, f, x)(t)⟩
          using prodFunctionApp group0_5_L1(2) by blast
        with assms st(2,3) have P(z) = ⟨x+s,x+t⟩ using group0_5_L2(2)
          by auto
        moreover
        from ‹x∈G› ‹s∈G› ‹t∈G› have (- (x+s)) + (x+t) = (-s)+t
          using cancel_middle_add(3) by simp
        with st2 have (- (x+s)) + (x+t) ∈ U by auto
        ultimately have P(z) ∈ {⟨s,t⟩ ∈ G × G . (- s) + t ∈ U}
          using assms st(2,3) group_op_closed by auto
        with as(3) have P(z) ∈ V by force
        with L z have z ∈ P-(V) using prodFunction func1_1_L5A vimage_iff
          by blast
      }
      with as(2) have ∃U∈𝒩₀. {⟨s,t⟩ ∈ G × G . (- s) + t ∈ U} ⊆ P-(V)

        by blast
      with ‹LeftTranslation(G,f,x):G→G› ‹V ⊆ G × G› have P-(V) ∈ leftUniformity
        unfolding leftUniformity_def using prodFunction func1_1_L6A by
blast
```

```
    } thus thesis by simp
  qed
  ultimately show thesis using IsUniformlyCont_def by auto
qed
```

Right translations are uniformly continuous with respect to the right uniformity.

```
lemma (in topgroup) right_mult_uniformity: assumes x∈G
  shows
    RightTranslation(G,f,x) {is uniformly continuous between} rightUniformity
{and} rightUniformity
proof -
  let P = ProdFunction(RightTranslation(G, f, x), RightTranslation(G, f,
x))
  from assms have R: RightTranslation(G,f,x):G→G and rightUniformity
{is a uniformity on} G
    using group0_5_L1 side_uniformities(2) by auto
  moreover have ∀V ∈ rightUniformity. P-(V) ∈ rightUniformity
  proof -
    { fix V assume V ∈ rightUniformity
      then obtain U where U ∈ 𝒩₀ and {⟨s,t⟩ ∈ G × G . s + (- t) ∈ U}
⊆ V
        unfolding rightUniformity_def by auto
      with ‹V ∈ rightUniformity› have
        as:V ⊆ G × G U ∈ 𝒩₀ {⟨s,t⟩ ∈ G × G . s + (- t) ∈ U} ⊆ V
        unfolding rightUniformity_def by auto
      { fix z assume z:z ∈ {⟨s,t⟩ ∈ G × G . s + (- t) ∈ U}
        then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
        from st(1) z have st2: s + (-t) ∈ U by auto
        from assms st have P(z) = ⟨RightTranslation(G, f, x)(s), RightTranslation(G,
f, x)(t)⟩
          using prodFunctionApp group0_5_L1(1) by blast
        with assms st(2,3) have P(z) = ⟨s+x,t+x⟩ using group0_5_L2(1)
          by auto
        moreover
        from ‹x∈G› ‹s∈G› ‹t∈G› have (s+x) + (-(t+x)) =s+ (-t)
          using cancel_middle_add(4) by simp
        with st2 have (s+x) + (-(t+x)) ∈ U by auto
        ultimately have P(z) ∈ {⟨s,t⟩ ∈ G × G . s + (- t) ∈ U}
          using assms st(2,3) group_op_closed by auto
        with as(3) have P(z) ∈ V by force
        with R z have z ∈ P-(V) using prodFunction func1_1_L5A vimage_iff
          by blast
      }
      with as(2) have ∃U∈𝒩₀. {⟨s,t⟩ ∈ G × G . s + (- t) ∈ U} ⊆ P-(V)

        by blast
      with ‹RightTranslation(G,f,x):G→G› ‹V ⊆ G × G› have P-(V) ∈ rightUniformity
        unfolding rightUniformity_def using prodFunction func1_1_L6A by
```

```
blast
    } thus thesis by simp
  qed
  ultimately show thesis using IsUniformlyCont_def by auto
qed
```

The third uniformity important on topological groups is called the uniformity of Roelcke.

```
definition(in topgroup) roelckeUniformity
 where roelckeUniformity ≡ {V∈Pow(G×G). ∃U∈ 𝒩₀. {⟨s,t⟩∈G×G. t ∈(U+s)+U}⊆
V}
```

The Roelcke uniformity is indeed a uniformity on the group.

```
lemma (in topgroup) roelcke_uniformity:
  shows roelckeUniformity {is a uniformity on} G
proof -
  let Φ = roelckeUniformity
  have ∀U ∈ Φ. id(G)⊆U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U) ∈ Φ
  proof
    fix U assume U:U ∈ roelckeUniformity
    then obtain V where V:{⟨s,t⟩∈G×G. t ∈(V+s)+V} ⊆ U V∈ 𝒩₀ U:Pow(G×G)

      unfolding roelckeUniformity_def by auto
    from V(2) have VG:V⊆G by auto
    have id(G) ⊆ U
    proof -
      from V(2) have 0∈int(V) by auto
      then have V0:0∈V using Top_2_L1 by auto
      { fix x assume x:x∈G
        with ‹V∈𝒩₀› have x ∈ V+x using elem_in_int_rtrans(1) Top_2_L1
by blast
        with ‹V⊆G› ‹x∈G› ‹0∈V› have x+0 : (V+x)+V
          using lrtrans_in_group_add(2) interval_add(4) by auto
        with ‹x∈G›  have x: (V+x)+V using group0_2_L2 by auto
        with ‹x∈G› have ⟨x,x⟩:{⟨s,t⟩∈G×G. t ∈(V+s)+V} by auto
        with V(1) have ⟨x,x⟩∈U by auto
      } thus id(G) ⊆ U by auto
    qed
    moreover have converse(U) ∈ Φ
    proof -
      { fix l assume l ∈ {⟨s,t⟩∈G×G. t ∈((-V)+s)+(-V)}
        then obtain s t where st:s∈G t∈G t ∈((-V)+s)+(-V) l=⟨s,t⟩
          by force
        from ‹V⊆G› have smG:(-V) ⊆ G using ginv_image_add(1) by simp
        with ‹s∈G› have VxG:(-V)+s ⊆ G using lrtrans_in_group_add(2)
by simp
        from ‹V⊆G› ‹t∈G› have VsG:V+t ⊆ G using lrtrans_in_group_add(2)
by simp
```

```
        from st(3) VxG smG obtain x y where xy:t = x+y x ∈ (-V)+s y∈(-
V)
          using elements_in_set_sum by blast
        from xy(2) smG st(1) obtain z where z:x = z+s z∈(-V) using elements_in_rtrans

          by blast
        with ‹y∈(-V)› ‹(-V)⊆G› ‹s∈G› ‹t = x+y›
        have ts:(-z)+t+(-y) = s using cancel_middle_add(5) by blast
        {
          fix u assume u∈(-V)
          with ‹V⊆G› have (-u) ∈ V using ginv_image_el_add by simp
        } hence R:∀u∈(-V).(-u) ∈ V by simp
        with z(2) xy(3) have zy:(-z)∈V (-y)∈V by auto
        from zy(1) VG st(2) have (-z)+t : V+t using lrtrans_image(2)
by auto
        with zy(2) VG VsG have (-z)+t+(-y) : (V+t)+V
          using interval_add(4) by auto
        with ts have s:(V+t)+V by auto
        with st(1,2) have ⟨s,t⟩ ∈ converse({⟨s,t⟩∈G×G. t ∈(V+s)+V})
          using converse_iff by auto
        with V(1) have ⟨s,t⟩ ∈  converse(U) by auto
        with st(4) have l ∈ converse(U) by auto
      } then have {⟨s,t⟩∈G×G. t ∈((-V)+s)+(-V)} ⊆ converse(U) by auto
      moreover from V(2) have (-V): 𝒩₀ using neg_neigh_neigh by auto
      ultimately have ∃V∈ 𝒩₀. {⟨s,t⟩∈G×G. t ∈(V+s)+V} ⊆ converse(U)
by auto
      moreover
      from V(3) have converse(U) ⊆ G×G unfolding converse_def by auto
      ultimately show converse(U) ∈ roelckeUniformity unfolding roelckeUniformity_def
by auto
    qed
    moreover have ∃Z∈Φ. Z O Z ⊆ U
    proof -
      from V(2) obtain W where W:W∈𝒩₀ W+W ⊆ V using exists_procls_zerohood
by blast
      then have WG:W⊆G by auto
      moreover
      { fix k assume as:k:{⟨s,t⟩∈G×G. t ∈(W+s)+W} O {⟨s,t⟩∈G×G. t ∈(W+s)+W}
        then obtain x₁ x₂ x₃ where
          x:x₁∈G x₂∈G x₃∈G x₂ ∈ (W+x₁)+W x₃ ∈ (W+x₂)+W k=⟨x₁,x₃⟩
          unfolding comp_def by auto
        from ‹x₁∈G› have VsG:W+x₁ ⊆ G and Vx1G:V+x₁ ⊆ G
          using lrtrans_in_group_add(2) by auto
        from x(4) VsG WG obtain x y where xy:x₂ = x+y x ∈ W+x₁ y∈W
          using elements_in_set_sum by blast
        from xy(2) WG x(1) obtain z where z:x = z+x₁ z∈W using elements_in_rtrans

          by blast
        from z(2) xy(3) WG have yzG:y∈G z∈G by auto
```

```
        from x(2) have VsG:W+x₂ ⊆ G using lrtrans_in_group_add by simp
        from x(5) VsG WG obtain x' y' where xy2:x₃ = x'+y' x'∈ W+x₂ y'∈W

          using elements_in_set_sum by blast
        from xy2(2) WG x(2) obtain z' where z2:x' = z'+x₂ z'∈W using
elements_in_rtrans
          by blast
        from z2(2) xy2(3) WG have yzG2:y'∈G z'∈G by auto
        from xy(1) z(1) xy2(1) z2(1) have x₃ = (z'+(z+x₁+y))+y' by auto
        with yzG yzG2 x(1) have x3:x₃ = ((z'+z)+x₁)+(y+y')
          using group_oper_assoc group_op_closed by simp
        from xy(3) z(2) xy2(3) z2(2) WG have z'+z ∈  W+W y+y' ∈ W+W
          using interval_add(4) by auto
        with W(2) have yzV:z'+z ∈ V y+y' ∈ V by auto
        from yzV(1) VG x(1) have (z'+z)+x₁ ∈ V+x₁ using lrtrans_image(2)
by auto
        with yzV(2) Vx1G VG have ((z'+z)+x₁)+(y+y') ∈ (V+x₁)+V
          using interval_add(4) by auto
        with x3 have x₃ ∈ (V+x₁)+V by auto
        with x(1,3,6) have k:{⟨s,t⟩∈G×G. t ∈(V+s)+V} by auto
      }
      with V(1) have {⟨s,t⟩∈G×G. t ∈(W+s)+W} O {⟨s,t⟩∈G×G. t ∈(W+s)+W}⊆U

        by auto
      moreover from W(1) have {⟨s,t⟩∈G×G. t ∈(W+s)+W}∈roelckeUniformity

        unfolding roelckeUniformity_def by auto
      ultimately show ∃Z∈roelckeUniformity. Z O Z ⊆ U by auto
    qed
    ultimately show id(G)⊆U ∧ (∃V∈Φ. V O V ⊆ U) ∧ converse(U) ∈ Φ
by simp
  qed
  moreover
  have roelckeUniformity {is a filter on} (G × G)
  proof -
    {
      assume 0 ∈ roelckeUniformity
      then obtain W where U:W∈𝒩₀ {⟨s,t⟩∈G×G. t ∈(W+s)+W}⊆0
        unfolding roelckeUniformity_def by auto
      have ⟨0,0⟩:G×G using zero_in_tgroup by auto
      moreover have 0 = 0+0+0 using group0_2_L2 zero_in_tgroup by auto
      moreover
      from U(1) have 0∈int(W) by auto
      then have 0∈W using Top_2_L1 by auto
      with ‹W∈𝒩₀› have 0+0+0 ∈ (W+0)+W
        using  group0_2_L2 group_op_closed trans_neutral_image interval_add_zero
        by auto
      ultimately have ⟨0,0⟩∈{⟨s,t⟩∈G×G. t ∈(W+s)+W} by auto
      with U(2) have False by blast
```

```
    }
    moreover
    {
      fix x xa assume as:x ∈ roelckeUniformity xa∈x
      have roelckeUniformity ⊆ Pow(G×G) unfolding roelckeUniformity_def
by auto
      with as have xa ∈ G×G by auto
    }
    moreover
    {
      have G×G∈Pow(G×G) by auto
      moreover
      have {⟨s,t⟩:G×G.  t ∈(G+s)+G} ⊆ G×G by auto
      moreover note zneigh_not_empty
      ultimately have G×G∈roelckeUniformity unfolding roelckeUniformity_def
by auto
    }
    moreover
    {
      fix A B assume as:A∈roelckeUniformity B∈roelckeUniformity
      from as(1) obtain AU where
        AU:AU∈ 𝒩₀ {⟨s,t⟩∈G×G. t ∈(AU+s)+AU}⊆ A A∈Pow(G×G)
        unfolding roelckeUniformity_def by auto
      from as(2) obtain BU where
        BU:BU∈ 𝒩₀ {⟨s,t⟩∈G×G. t ∈(BU+s)+BU}⊆ B B∈Pow(G×G)
        unfolding roelckeUniformity_def by auto
      from AU(1) BU(1) have 0 ∈ int(AU)∩int(BU) by auto
      moreover have op:int(AU)∩int(BU)∈T using Top_2_L2 topSpaceAssum
unfolding IsATopology_def
        by auto
      moreover
      have int(AU)∩int(BU) ⊆ AU∩BU using Top_2_L1 by auto
      with op have int(AU)∩int(BU)⊆int(AU∩BU) using Top_2_L5
        by auto
      moreover note AU(1) BU(1)
      ultimately have interNeigh:AU∩BU ∈ 𝒩₀ unfolding zerohoods_def by
auto
      moreover
      {
        fix z assume z ∈ {⟨s,t⟩∈G×G. t ∈((AU∩BU)+s)+(AU∩BU)}
        then obtain s t where
          z:z=⟨s,t⟩ s∈G t∈G t ∈ ((AU∩BU)+s)+(AU∩BU)
          by force
        from ‹AU∩BU ∈ 𝒩₀› ‹s∈G› have AU∩BU ⊆ G and (AU∩BU)+s ⊆G
          using lrtrans_in_group_add(2) by auto
        with z(4) obtain  x y where t:t=x+y x∈(AU∩BU)+s y∈AU∩BU
          using elements_in_set_sum by blast
        from t(2) z(2) interNeigh obtain q where x:x=q+s q ∈ AU∩BU us-
ing lrtrans_image(2)
```

```
          by auto
        with AU(1) BU(1) z(2) have x ∈ AU+s x ∈ BU+s using lrtrans_image(2)
by auto
        with ‹y ∈ AU∩BU› ‹AU∈ 𝒩₀› ‹BU∈ 𝒩₀› ‹s ∈ G› ‹t=x+y› have
          t ∈ (AU+s)+AU and t ∈ (BU+s)+BU
          using lrtrans_in_group_add(2) elements_in_set_sum_inv by auto

        with z(1,2,3) have
          z ∈ {⟨s,t⟩∈G×G. t ∈(AU+s)+AU} and z ∈{⟨s,t⟩∈G×G. t ∈(BU+s)+BU}

          by auto
      }
      then have
        {⟨s,t⟩∈G×G. t ∈((AU∩BU)+s)+(AU∩BU)} ⊆
        {⟨s,t⟩∈G×G. t ∈(AU+s)+AU}∩{⟨s,t⟩∈G×G. t ∈(BU+s)+BU}
        by auto
      with AU(2) BU(2) have {⟨s,t⟩∈G×G. t ∈((AU∩BU)+s)+(AU∩BU)}⊆A∩B
by blast
      ultimately have A∩B ∈ roelckeUniformity using AU(3) BU(3) unfold-
ing roelckeUniformity_def
        by blast
    }
    moreover
    {
      fix B C assume as:B∈roelckeUniformity C⊆(G × G) B ⊆ C
      from as(1) obtain BU where BU:BU∈𝒩₀ {⟨s,t⟩∈G×G. t ∈(BU+s)+BU}⊆B

        unfolding roelckeUniformity_def by blast
      from as(3) BU(2) have {⟨s,t⟩∈G×G. t ∈(BU+s)+BU}⊆C by auto
      then have C ∈ roelckeUniformity using as(2) BU(1) unfolding roelckeUniformity_def

        by auto
    }
    ultimately show thesis unfolding IsFilter_def by auto
  qed
  ultimately show thesis using IsUniformity_def by auto
qed
```

The topology given by the roelcke uniformity is the original topology

```
lemma (in topgroup) top_generated_roelcke_uniformity:
  shows UniformTopology(roelckeUniformity,G) = T
proof -
  let M = {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩ . t ∈ G}
  have fun:M:G→Pow(Pow(G)) using IsNeighSystem_def neigh_from_uniformity
roelcke_uniformity
    by auto
  {
    fix U assume as:U ∈ {U ∈ Pow(G). ∀x∈U. U ∈ Mx}
    {
```

```
      fix x assume x:x∈U
      with as have xg:x∈G by auto
      from x as have U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩ . t ∈
G}(x) by auto
      with fun ‹x∈G› have U ∈ {V  {x} . V ∈ roelckeUniformity} using
ZF_fun_from_tot_val
        by simp
      then obtain V where V:U=V{x} V ∈ roelckeUniformity by auto
      from V(2) obtain W where W:W∈𝒩₀ {⟨s,t⟩∈G×G. t∈ (W+s)+W}⊆V
        unfolding roelckeUniformity_def by auto
      from W(1) have WG:W⊆G by auto
      from W(2) have A:{⟨s,t⟩:G×G. t:(W+s)+W}{x} ⊆ V{x} by auto
      have {⟨s,t⟩ ∈ G×G. t ∈ (W+s)+W}{x} = (W+x)+W
      proof -
        let A = {⟨s,t⟩:G×G. t ∈ (W+s)+W}
        from ‹W⊆G›  ‹x∈G› have I:(W+x)+W ⊆ G
            using  lrtrans_in_group_add interval_add(1) by auto
        have A{x} = {t∈G.⟨x,t⟩ ∈ A} by blast
        moreover have {t∈G. ⟨x,t⟩ ∈ A} ⊆ (W+x)+W by auto
        moreover from ‹W⊆G›  ‹x∈G› I have (W+x)+W ⊆ {t∈G. ⟨x,t⟩ ∈ A}
by auto
        ultimately show thesis by auto
      qed
      with A V(1) have WU:(W+x)+W ⊆ U by auto
      have int(W)+x ⊆ W+x using image_mono Top_2_L1 by simp
      then have (int(W)+x)×(int(W)) ⊆ (W+x)×W using Top_2_L1 by auto
      then have f((int(W)+x)×(int(W))) ⊆ f((W+x)×W) using image_mono
by auto
      moreover
      from xg WG have
        ⟨int(W)+x,int(W)⟩ ∈ Pow(G)×Pow(G) and ⟨(W+x),W⟩ ∈ Pow(G)×Pow(G)

        using Top_2_L2 lrtrans_in_group_add(2) by auto
      then have
        (int(W)+x)+(int(W)) = f((int(W)+x)×(int(W))) and
        (W+x)+W = f((W+x)×W)
        using interval_add(2) by auto
      ultimately have (int(W)+x)+(int(W)) ⊆ (W+x)+W by auto
      with xg WG have int(W+x)+(int(W)) ⊆ (W+x)+W using rtrans_interior

        by auto
      moreover
      {
        have int(W+x)+(int(W)) = (⋃t∈int(W+x). t+(int(W)))
          using interval_add(3) Top_2_L2 by auto
        moreover have ∀t∈int(W+x). t+(int(W)) = int(t+W)
        proof -
          { fix t assume t ∈ int(W+x)
            from ‹x∈G› have (W+x) ⊆ G using lrtrans_in_group_add(2)
```

```
by simp
            with ‹t ∈ int(W+x)› have t∈G using Top_2_L2 by auto
            with ‹W⊆G› have t + int(W) = int(t+W) using ltrans_interior
by simp
          } thus thesis by simp
        qed
        ultimately have int(W+x)+(int(W)) = (⋃t∈int(W+x). int(t+W))
          by auto
        with topSpaceAssum have int(W+x)+(int(W)) ∈ T using Top_2_L2
union_open
          by auto
      }
      moreover from ‹x∈G› ‹W∈𝒩₀› have x ∈ int(W+x)+(int(W))
        using elem_in_int_rtrans(2) by simp
      moreover note WU
      ultimately have ∃Y∈T. x∈Y ∧ Y⊆U by auto
    }
    then have U∈T using open_neigh_open by auto
  }
  then have {U ∈ Pow(G). ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩
. t ∈ G}(x)} ⊆ T
    by auto
  moreover
  {
    fix U assume op:U∈T
    {
      fix x assume x:x∈U
      with op have xg:x∈G by auto
      have (-x)+U ∈ 𝒩₀ using open_trans_neigh op x by auto
      then obtain W where W:W∈𝒩₀ W + W ⊆(-x)+U using exists_procls_zerohood

        by blast
      let V = x+(W+(-x)) ∩ W
      from ‹W ∈ 𝒩₀› ‹x∈G› have  xWx:x+(W+(-x)) :𝒩₀ using lrtrans_neigh
        by simp
      from W(1) have WG:W⊆G by auto
      from xWx W(1) have 0∈int(x+(W+(-x)))∩int(W) by auto
      have int:int(x+(W+(-x)))∩int(W)∈T
        using Top_2_L2 topSpaceAssum  unfolding IsATopology_def by auto
      have int(x+(W+(-x)))∩int(W) ⊆ (x+(W+(-x)))∩W using Top_2_L1
        by auto
      with int have int(x+(W+(-x)))∩int(W)⊆int((x+(W+(-x)))∩W)
        using Top_2_L5 by auto
      moreover note xWx W(1)
      ultimately have V_NEIG:V ∈ 𝒩₀ unfolding zerohoods_def by auto
      {
        fix z assume z:z ∈ (V+x)+V
        from W(1) have VG:V ⊆ G by auto
        with ‹x∈G› have VxG:V+x ⊆ G using lrtrans_in_group_add(2) by
```

```
simp
        from z VG VxG W(1) obtain a₁ b₁ where ab:z=a₁+b₁ a₁ ∈ V+x b₁
∈ V
          using elements_in_set_sum by blast
        from ab(2) xg VG obtain c₁ where c:a₁=c₁+x c₁ ∈ V using elements_in_rtrans

          by blast
        from ab(3) c(2) have bc:b₁∈W c₁ ∈ x+(W+(-x)) by auto
        from ‹x∈G› have x+(W+(-x)) = {x+y. y∈(W+(-x))}
          using neg_in_tgroup lrtrans_in_group_add lrtrans_image by auto
        with ‹c₁ ∈ x+(W+(-x))› obtain d where d:c₁=x+d d ∈ W+(-x)
          by auto
        from ‹x∈G› ‹W∈𝒩₀› ‹d ∈ W+(-x)› obtain e where e:d=e+(-x) e∈W
          using neg_in_tgroup lrtrans_in_group lrtrans_image(2) by auto

        from e(2) WG have eG:e∈G by auto
        from ‹e∈W› ‹W⊆G› ‹b₁∈W› have e∈G b₁∈G by auto
        from ‹z = a₁+b₁› ‹a₁ = c₁+x›  ‹c₁ = x+d› ‹d = e+(-x)›
        have z = x + (e+(-x)) + x + b₁ by simp
        with ‹x∈G› ‹e∈G› have z=(x+e)+b₁ using cancel_middle(4) by simp
        with ‹x∈G› ‹e∈G› ‹b₁∈G› have z = x+(e+b₁) using group_oper_assoc
by simp
        moreover from e(2) ab(3) WG have e+b₁ ∈ W+W using elements_in_set_sum_inv

          by auto
        moreover note xg WG
        ultimately have z∈x+(W+W) using elements_in_ltrans_inv interval_add(1)

          by auto
        moreover
        from ‹W⊆G› ‹U∈T› have W + W ⊆ G and U⊆G using interval_add(1)
by auto
        with ‹W + W ⊆(-x)+U› ‹x∈G› have  x+(W+W) ⊆ U using trans_subset

          by simp
        ultimately have z∈U by auto
      }
      then have sub:(V+x)+V ⊆ U by auto
      moreover from V_NEIG have unif:{⟨s,t⟩ ∈ G×G. t : (V+s)+V} ∈ roelckeUniformity

        unfolding roelckeUniformity_def by auto
      moreover from xg have
        ∀q. q∈{⟨s,t⟩ ∈ G×G. t : (V+s)+V}{x} ⟷ q∈((V+x)+V)∩G
        by auto
      then have {⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V}{x} = ((V+x)+V)∩G
        by auto
      ultimately have basic:{⟨s,t⟩ ∈ G×G. t : (V+s)+V}{x} ⊆ U using op

        by auto
```

```
      have add:({x}×U){x} =U by auto
      from basic add have ({⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V}∪({x}×U)){x} =
U
        by auto
      moreover have R:∀B∈roelckeUniformity.(∀ C∈Pow(G × G).  B ⊆ C
⟶ C ∈ roelckeUniformity)
        using roelcke_uniformity unfolding IsUniformity_def IsFilter_def
by auto
      moreover from op xg have GG:({⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V}∪({x}×U)):Pow(G×G)
by auto
      moreover have
        {⟨s,t⟩ ∈ G×G. t∈(V+s)+V} ⊆ ({⟨s,t⟩ ∈ G×G. t∈(V+s)+V}∪({x}×U))

        by auto
      moreover from R unif GG have
        ({⟨s,t⟩ ∈ G×G. t ∈ (V+s)+V}∪({x}×U)) ∈ roelckeUniformity by
auto
      ultimately have ∃V∈roelckeUniformity. V{x} = U by auto
      then have U ∈ {V  {x} . V ∈ roelckeUniformity} by auto
      with xg fun have U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩ . t
∈ G}  x
        using apply_equality by auto
    } hence ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩ . t ∈ G}
x by auto
    with op have U:{U ∈ Pow(G). ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩
. t ∈ G}(x)}
      by auto
  } then have T ⊆ {U ∈ Pow(G). ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩
. t ∈ G}(x)}
    by auto
  ultimately have
    {U ∈ Pow(G). ∀x∈U. U ∈ {⟨t, {V  {t} . V ∈ roelckeUniformity}⟩ . t
∈ G}(x)} = T by auto
  then show thesis using uniftop_def_alt by simp
qed
```

The inverse map is uniformly continuous in the Roelcke uniformity

```
theorem (in topgroup) inv_uniform_roelcke:
  shows
    GroupInv(G,f) {is uniformly continuous between} roelckeUniformity
{and} roelckeUniformity
proof -
  let P = ProdFunction(GroupInv(G,f), GroupInv(G,f))
  have L: GroupInv(G,f):G→G and R:roelckeUniformity {is a uniformity
on} G
    using groupAssum group0_2_T2 roelcke_uniformity by auto
  have ∀V ∈ roelckeUniformity. P-(V) ∈ roelckeUniformity
  proof
    fix V assume v:V∈ roelckeUniformity
```

```
    then obtain U where U ∈ 𝒩0 and {⟨s,t⟩ ∈ G × G . t ∈ U + s + U}
⊆ V
      unfolding roelckeUniformity_def by auto
    with ‹V ∈ roelckeUniformity› have
      as:V ⊆ G × G U ∈ 𝒩0 {⟨s,t⟩ ∈ G × G . t ∈ U + s + U} ⊆ V
      unfolding roelckeUniformity_def by auto
    from as(2) obtain W where w:W ∈ 𝒩0 W ⊆ U (-W) = W using exists_sym_zerohood
by blast
    from w(1) have wg:W⊆G by auto
    {
      fix z assume z:z ∈ {⟨s,t⟩ ∈ G × G . t ∈ W + s + W}
      then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
      from st(1) z have st2: t ∈ W + s + W by auto
      with ‹W ∈ 𝒩0› st(2) obtain u v where uv:t=u+v u∈W+s v∈W
        using interval_add(4) lrtrans_in_group_add(2)  by auto
      from ‹W⊆G› ‹s∈G› ‹u∈W+s› obtain q where q:q∈W u=q+s using elements_in_rtrans

        by blast
      from w(2) as(2) q st(2) have u∈U+s using lrtrans_image(2) by auto
      with w(2) uv(1,3) as(2) st(2) have t∈U + s + U using interval_add(4)
        lrtrans_in_group_add(2) by auto
      with st have z ∈ {⟨s,t⟩ ∈ G × G . t ∈ U + s + U} by auto
    }
    then have
      sub:{⟨s,t⟩ ∈ G × G . t ∈ W + s + W} ⊆ {⟨s,t⟩ ∈ G × G . t ∈ U +
s + U}
      by auto
    {
      fix z assume z:z ∈ {⟨s,t⟩ ∈ G × G . t ∈ W + s + W}
      then obtain s t where st:z=⟨s,t⟩ s∈G t∈G by auto
      from st(1) z have st2: t ∈ W + s + W by auto
      with ‹W ∈ 𝒩0› obtain u v where uv:t=u+v u∈W+s v∈W
        using interval_add(4) lrtrans_in_group_add(2) st(2) by auto
      from ‹W⊆G› ‹s∈G› ‹u∈W+s› obtain q where q:q∈W u=q+s using elements_in_rtrans

        by blast
      from ‹W⊆G› ‹q∈W› ‹v∈W› have q∈G v∈G by auto
      with ‹q∈G› ‹v∈G› ‹u=q+s› st(2) uv(1) q(2) have t=q+(s+v)
        using group_op_closed_add group_oper_assoc by auto
      with st(2) ‹q∈G› ‹v∈G› have minust:(-t) = (-v)+(-s)+(-q)
        using group_inv_of_two group_op_closed group_inv_of_two by auto
      from q(1) wg have (-q)∈-W using ginv_image_add(2) by auto
      with w(3) have minusq:(-q)∈W by auto
      from uv(3) wg have (-v)∈-W using ginv_image_add(2) by auto
      with w(3) have minusv:(-v)∈W by auto
      with st(2) wg have (-v)+(-s) ∈ W+(-s)
        using lrtrans_image(2) inverse_in_group by auto
      with minust minusq st(2) wg have (-t)∈(W+(-s))+W
        using interval_add(4) inverse_in_group lrtrans_in_group_add(2)
```

```
by auto
      moreover
      from st groupAssum have P(z) = ⟨GroupInv(G,f)(s), GroupInv(G,f)(t)⟩
        using prodFunctionApp group0_2_T2 by blast
      with st(2,3) have P(z) = ⟨-s,-t⟩ by auto
      ultimately have P(z) ∈ {⟨s,t⟩ ∈ G × G . t ∈ W + s + W}
        using st(2,3) inverse_in_group by auto
      with sub have P(z) ∈ {⟨s,t⟩ ∈ G × G . t ∈ U + s + U} by force
      with as(3) have P(z) ∈ V by force
      with z L have z ∈ P-(V) using prodFunction func1_1_L5A vimage_iff
        by blast
    }
    with w(1) have ∃U∈𝒩₀. {⟨s,t⟩ ∈ G × G . t ∈ U + s + U} ⊆ P-(V)
      by blast
    with L show P-(V) ∈ roelckeUniformity
      unfolding roelckeUniformity_def using prodFunction func1_1_L6A by
blast
  qed
  with L R show thesis using IsUniformlyCont_def by auto
qed

end
```

100 Topological groups 2

```
theory TopologicalGroup_ZF_2 imports Topology_ZF_8 TopologicalGroup_ZF
Group_ZF_2
begin
```

This theory deals with quotient topological groups.

100.1 Quotients of topological groups

The quotient topology given by the quotient group equivalent relation, has an open quotient map.

```
theorem(in topgroup) quotient_map_topgroup_open:
  assumes IsAsubgroup(H,f) A∈T
  defines r ≡ QuotientGroupRel(G,f,H)
  shows {⟨b,r{b}⟩. b∈⋃T}A∈(T{quotient by}r)
proof-
  have eqT:equiv(⋃T,r) and eqG:equiv(G,r) using group0.Group_ZF_2_4_L3
assms(1) unfolding r_def IsAnormalSubgroup_def
    using group0_valid_in_tgroup by auto
  have subA:A⊆G using assms(2) by auto
  have subH:H⊆G using group0.group0_3_L2[OF group0_valid_in_tgroup assms(1)].
  have A1:{⟨b,r{b}⟩. b∈⋃T}-({⟨b,r{b}⟩. b∈⋃T}A)=H+A
  proof
    {
```

```
      fix t assume t∈{⟨b,r{b}⟩. b∈⋃T}-({⟨b,r{b}⟩. b∈⋃T}A)
      then have ∃m∈({⟨b,r{b}⟩. b∈⋃T}A). ⟨t,m⟩∈{⟨b,r{b}⟩. b∈⋃T} using
vimage_iff by auto
      then obtain m where m∈({⟨b,r{b}⟩. b∈⋃T}A)⟨t,m⟩∈{⟨b,r{b}⟩. b∈⋃T}
by auto
      then obtain b where b∈A⟨b,m⟩∈{⟨b,r{b}⟩. b∈⋃T}t∈G and rel:r{t}=m
using image_iff by auto
      then have r{b}=m by auto
      then have r{t}=r{b} using rel by auto
      with ‹b∈A›subA have ⟨t,b⟩∈r using eq_equiv_class[OF _ eqT] by auto
      then have f⟨t,GroupInv(G,f)b⟩∈H unfolding r_def QuotientGroupRel_def
by auto
      then obtain h where h∈H and prd:f⟨t,GroupInv(G,f)b⟩=h by auto
      then have h∈G using subH by auto
      have b∈G using ‹b∈A›‹A∈T› by auto
      then have (-b)∈G using neg_in_tgroup by auto
      from prd have h=t+(-b) by simp
      with ‹t∈G› ‹b∈G› have t = h+b using inv_cancel_two_add(1) by simp

      then have ⟨⟨h,b⟩,t⟩∈f using apply_Pair[OF topgroup_f_binop] ‹h∈G›‹b∈G›
by auto
      moreover from ‹h∈H›‹b∈A› have ⟨h,b⟩∈H×A by auto
      ultimately have t∈f(H×A) using image_iff by auto
      with subA subH have t∈H+A using interval_add(2) by auto
    }
    then show ({⟨b,r{b}⟩. b∈⋃T}-({⟨b,r{b}⟩. b∈⋃T}A))⊆H+A by force
    {
      fix t assume t∈H+A
      with subA subH have t ∈ f(H×A) using interval_add(2) by auto
      then obtain ha where ha∈H×A ⟨ha,t⟩∈f using image_iff by auto
      then obtain h aa where ha=⟨h,aa⟩h∈Haa∈A by auto
      then have h∈Gaa∈G using subH subA by auto
      from ‹⟨ha,t⟩∈f› have t∈G using topgroup_f_binop unfolding Pi_def
by auto
      from ‹ha=⟨h,aa⟩› ‹⟨ha,t⟩∈f› have t=h+aa using apply_equality topgroup_f_binop

        by auto
      with ‹h∈G› ‹aa∈G› have t+(-aa) = h using inv_cancel_two_add(2)
by simp
      with ‹h∈H›‹t∈G›‹aa∈G› have ⟨t,aa⟩∈r unfolding r_def QuotientGroupRel_def
by auto
      then have r{t}=r{aa} using eqT equiv_class_eq by auto
      with ‹aa∈G› have ⟨aa,r{t}⟩∈{⟨b,r{b}⟩. b∈⋃T} by auto
      with ‹aa∈A› have A1:r{t}∈({⟨b,r{b}⟩. b∈⋃T}A) using image_iff by
auto
      from ‹t∈G› have ⟨t,r{t}⟩∈{⟨b,r{b}⟩. b∈⋃T} by auto
      with A1 have t∈{⟨b,r{b}⟩. b∈⋃T}-({⟨b,r{b}⟩. b∈⋃T}A) using vimage_iff
by auto
    }
```

```
    then show H+A⊆{⟨b,r{b}⟩. b∈⋃T}-({⟨b,r{b}⟩. b∈⋃T}A) by auto
  qed
  have H+A=(⋃x∈H. x + A) using interval_add(3) subH subA by auto moreover
  have ∀x∈H. x + A∈T using open_tr_open(1) assms(2) subH by blast
  then have {x + A. x∈H}⊆T by auto
  then have (⋃x∈H. x + A)∈T using topSpaceAssum unfolding IsATopology_def
by auto
  ultimately have H+A∈T by auto
  with A1 have {⟨b,r{b}⟩. b∈⋃T}-({⟨b,r{b}⟩. b∈⋃T}A)∈T by auto
  then have ({⟨b,r{b}⟩. b∈⋃T}A)∈{quotient topology in}((⋃T)//r){by}{⟨b,r{b}⟩.
b∈⋃T}{from}T
    using QuotientTop_def topSpaceAssum quotient_proj_surj using
    func1_1_L6(2)[OF quotient_proj_fun] by auto
  then show ({⟨b,r{b}⟩. b∈⋃T}A)∈(T{quotient by}r) using EquivQuo_def[OF
eqT] by auto
qed
```

A quotient of a topological group is just a quotient group with an appropiate topology that makes product and inverse continuous.

```
theorem (in topgroup) quotient_top_group_F_cont:
  assumes IsAnormalSubgroup(G,f,H)
  defines r ≡ QuotientGroupRel(G,f,H)
  defines F ≡ QuotientGroupOp(G,f,H)
  shows IsContinuous(ProductTopology(T{quotient by}r,T{quotient by}r),T{quotient
by}r,F)
proof-
  have eqT:equiv(⋃T,r) and eqG:equiv(G,r) using group0.Group_ZF_2_4_L3
assms(1) unfolding r_def IsAnormalSubgroup_def
    using group0_valid_in_tgroup by auto
  have fun:{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}:G×G→(G//r)×(G//r) using product_equiv_rel_fun unfolding G_def by auto
  have C:Congruent2(r,f) using Group_ZF_2_4_L5A[OF Ggroup assms(1)] unfolding r_def.
  with eqT have IsContinuous(ProductTopology(T,T),ProductTopology(T{quotient
by}r,T{quotient by}r),{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T})
    using product_quo_fun by auto
  have tprod:topology0(ProductTopology(T,T)) unfolding topology0_def using Top_1_4_T1(1)[OF topSpaceAssum topSpaceAssum].
  have Hfun:{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}∈surj(⋃ProductTopology(T,T),⋃(({quotient
topology in}(((⋃T)//r)×((⋃T)//r)){by}{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}{from}(ProductTopo
using prod_equiv_rel_surj
    total_quo_equi[OF eqT] topology0.total_quo_func[OF tprod prod_equiv_rel_surj]
unfolding F_def QuotientGroupOp_def r_def
    by auto
  have Ffun:F:⋃(({quotient topology in}(((⋃T)//r)×((⋃T)//r)){by}{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩.
⟨b,c⟩∈⋃T×⋃T}{from}(ProductTopology(T,T))))→⋃(T{quotient by}r)
    using EquivClass_1_T1[OF eqG C] using total_quo_equi[OF eqT] topology0.total_quo_func[O
tprod prod_equiv_rel_surj] unfolding F_def QuotientGroupOp_def r_def
```

```
    by auto
  have cc:(F O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}):G×G→G//r using comp_fun[OF
fun EquivClass_1_T1[OF eqG C]]
    unfolding F_def QuotientGroupOp_def r_def by auto
  then have (F O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}):⋃(ProductTopology(T,T))→⋃(T{quotient
by}r) using Top_1_4_T1(3)[OF topSpaceAssum topSpaceAssum]
    total_quo_equi[OF eqT] by auto
  then have two:two_top_spaces0(ProductTopology(T,T),T{quotient by}r,(F
O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T})) unfolding two_top_spaces0_def
    using Top_1_4_T1(1)[OF topSpaceAssum topSpaceAssum] equiv_quo_is_top[OF
eqT] by auto
  have IsContinuous(ProductTopology(T,T),T,f) using fcon prodtop_def by
auto moreover
  have IsContinuous(T,T{quotient by}r,{⟨b,r{b}⟩. b∈⋃T}) using quotient_func_cont[OF
quotient_proj_surj]
    unfolding EquivQuo_def[OF eqT] by auto
  ultimately have cont:IsContinuous(ProductTopology(T,T),T{quotient by}r,{⟨b,r{b}⟩.
b∈⋃T} O f)
    using comp_cont by auto
  {
    fix A assume A:A∈G×G
    then obtain g1 g2 where A_def:A=⟨g1,g2⟩ g1∈Gg2∈G by auto
    then have fA=g1+g2 and p:g1+g2∈⋃T unfolding grop_def using
      apply_type[OF topgroup_f_binop] by auto
    then have {⟨b,r{b}⟩. b∈⋃T}(fA)={⟨b,r{b}⟩. b∈⋃T}(g1+g2) by auto
    with p have {⟨b,r{b}⟩. b∈⋃T}(fA)=r{g1+g2} using apply_equality[OF
_ quotient_proj_fun]
      by auto
    then have Pr1:({⟨b,r{b}⟩. b∈⋃T} O f)A=r{g1+g2} using comp_fun_apply[OF
topgroup_f_binop A] by auto
    from A_def(2,3) have ⟨g1,g2⟩∈⋃T×⋃T by auto
    then have ⟨⟨g1,g2⟩,⟨r{g1},r{g2}⟩⟩∈{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}
by auto
    then have {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}A=⟨r{g1},r{g2}⟩ using
A_def(1) apply_equality[OF _ product_equiv_rel_fun]
      by auto
    then have F({⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}A)=F⟨r{g1},r{g2}⟩ by
auto
    then have F({⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}A)=r({g1+g2}) using
group0.Group_ZF_2_2_L2[OF group0_valid_in_tgroup eqG C
      _ A_def(2,3)] unfolding F_def QuotientGroupOp_def r_def by auto
moreover
    note fun ultimately have (F O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T})A=r({g1+g2})
using comp_fun_apply[OF _ A] by auto
    then have (F O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T})A=({⟨b,r{b}⟩. b∈⋃T}
O f)A using Pr1 by auto
  }
  then have (F O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T})=({⟨b,r{b}⟩. b∈⋃T}
O f) using fun_extension[OF cc comp_fun[OF topgroup_f_binop quotient_proj_fun]]
```

```
    unfolding F_def QuotientGroupOp_def r_def by auto
  then have A:IsContinuous(ProductTopology(T,T),T{quotient by}r,F O {⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩.
⟨b,c⟩∈⋃T×⋃T}) using cont by auto
  have IsAsubgroup(H,f) using assms(1) unfolding IsAnormalSubgroup_def
by auto
  then have ∀A∈T. {⟨b, r  {b}⟩ . b ∈ ⋃T}  A ∈ ({quotient by}r) using
quotient_map_topgroup_open unfolding r_def by auto
  with eqT have ProductTopology({quotient by}r,{quotient by}r)=({quotient
topology in}(((⋃T)//r)×((⋃T)//r)){by}{⟨⟨b,c⟩,⟨r{b},r{c}⟩⟩. ⟨b,c⟩∈⋃T×⋃T}{from}(ProductTopo
using prod_quotient
    by auto
  with A show IsContinuous(ProductTopology(T{quotient by}r,T{quotient
by}r),T{quotient by}r,F)
    using two_top_spaces0.cont_quotient_top[OF two Hfun Ffun] topology0.total_quo_func[OF
tprod prod_equiv_rel_surj] unfolding F_def QuotientGroupOp_def r_def
    by auto
qed

lemma (in group0) Group_ZF_2_4_L8:
  assumes IsAnormalSubgroup(G,P,H)
  defines r ≡ QuotientGroupRel(G,P,H)
  and F ≡ QuotientGroupOp(G,P,H)
  shows GroupInv(G//r,F):G//r→G//r
  using group0_2_T2[OF Group_ZF_2_4_T1[OF _ assms(1)]] groupAssum us-
ing assms(2,3)
    by auto

theorem (in topgroup) quotient_top_group_INV_cont:
  assumes IsAnormalSubgroup(G,f,H)
  defines r ≡ QuotientGroupRel(G,f,H)
  defines F ≡ QuotientGroupOp(G,f,H)
  shows IsContinuous(T{quotient by}r,T{quotient by}r,GroupInv(G//r,F))
proof-
  have eqT:equiv(⋃T,r) and eqG:equiv(G,r) using group0.Group_ZF_2_4_L3
assms(1) unfolding r_def IsAnormalSubgroup_def
    using group0_valid_in_tgroup by auto
  have two:two_top_spaces0(T,T{quotient by}r,{⟨b,r{b}⟩. b∈G}) unfold-
ing two_top_spaces0_def
    using topSpaceAssum equiv_quo_is_top[OF eqT] quotient_proj_fun total_quo_equi[OF
eqT] by auto
  have IsContinuous(T,T,GroupInv(G,f)) using inv_cont. moreover
  {
    fix g assume G:g∈G
    then have GroupInv(G,f)g=-g using grinv_def by auto
    then have r({GroupInv(G,f)g})=GroupInv(G//r,F)(r{g}) using group0.Group_ZF_2_4_L7
      [OF group0_valid_in_tgroup assms(1) G] unfolding r_def F_def by
auto
    then have {⟨b,r{b}⟩. b∈G}(GroupInv(G,f)g)=GroupInv(G//r,F)({⟨b,r{b}⟩.
b∈G}g)
```

```
      using apply_equality[OF _ quotient_proj_fun] G neg_in_tgroup unfolding grinv_def
      by auto
    then have ({⟨b,r{b}⟩. b∈G}O GroupInv(G,f))g=(GroupInv(G//r,F)O {⟨b,r{b}⟩. b∈G})g
      using comp_fun_apply[OF quotient_proj_fun G] comp_fun_apply[OF group0_2_T2[OF Ggroup] G] by auto
  }
  then have A1:{⟨b,r{b}⟩. b∈G}O GroupInv(G,f)=GroupInv(G//r,F)O {⟨b,r{b}⟩. b∈G} using fun_extension[
    OF comp_fun[OF quotient_proj_fun group0.Group_ZF_2_4_L8[OF group0_valid_in_tgroup assms(1)]]
    comp_fun[OF group0_2_T2[OF Ggroup] quotient_proj_fun[of Gr]]] unfolding r_def F_def by auto
  have IsContinuous(T,T{quotient by}r,{⟨b,r{b}⟩. b∈⋃T}) using quotient_func_cont[OF quotient_proj_surj]
    unfolding EquivQuo_def[OF eqT] by auto
  ultimately have IsContinuous(T,T{quotient by}r,{⟨b,r{b}⟩. b∈⋃T}O GroupInv(G,f))
    using comp_cont by auto
  with A1 have IsContinuous(T,T{quotient by}r,GroupInv(G//r,F)O {⟨b,r{b}⟩. b∈G}) by auto
  then have IsContinuous({quotient topology in}(⋃T) // r{by}{⟨b, r  {b}⟩ . b ∈ ⋃T}{from}T,T{quotient by}r,GroupInv(G//r,F))
    using two_top_spaces0.cont_quotient_top[OF two quotient_proj_surj, of GroupInv(G//r,F)r] group0.Group_ZF_2_4_L8[OF group0_valid_in_tgroup assms(1)]
    using total_quo_equi[OF eqT] unfolding r_def F_def by auto
  then show thesis unfolding EquivQuo_def[OF eqT].
qed
```

Finally we can prove that quotient groups of topological groups are topological groups.

```
theorem(in topgroup) quotient_top_group:
  assumes IsAnormalSubgroup(G,f,H)
  defines r ≡ QuotientGroupRel(G,f,H)
  defines F ≡ QuotientGroupOp(G,f,H)
  shows IsAtopologicalGroup({quotient by}r,F)
    unfolding IsAtopologicalGroup_def using total_quo_equi equiv_quo_is_top
    Group_ZF_2_4_T1 Ggroup assms(1) quotient_top_group_INV_cont quotient_top_group_F_cont
    group0.Group_ZF_2_4_L3 group0_valid_in_tgroup assms(1) unfolding r_def F_def IsAnormalSubgroup_def
    by auto


end
```

101 Topological groups 3

```
theory TopologicalGroup_ZF_3 imports Topology_ZF_10 TopologicalGroup_ZF_2
TopologicalGroup_ZF_1
  Group_ZF_4
```

begin

This theory deals with topological properties of subgroups, quotient groups and relations between group theorical properties and topological properties.

101.1 Subgroups topologies

The closure of a subgroup is a subgroup.

```
theorem (in topgroup) closure_subgroup:
  assumes IsAsubgroup(H,f)
  shows IsAsubgroup(cl(H),f)
proof-
  have two:two_top_spaces0(ProductTopology(T,T),T,f) unfolding two_top_spaces0_def
using
    topSpaceAssum Top_1_4_T1(1,3) topgroup_f_binop by auto
  from fcon have cont:IsContinuous(ProductTopology(T,T),T,f) by auto
  then have closed:∀D. D{is closed in}T ⟶ f-D{is closed in}τ using
two_top_spaces0.TopZF_2_1_L1
    two by auto
  then have closure:∀A∈Pow(⋃τ). f(Closure(A,τ))⊆cl(fA) using two_top_spaces0.Top_ZF_2_1_L
    two by force
  have sub1:H⊆G using group0.group0_3_L2 group0_valid_in_tgroup assms
by force
  then have sub:(H)×(H)⊆⋃τ using prod_top_on_G(2) by auto
  from sub1 have clHG:cl(H)⊆G using Top_3_L11(1) by auto
  then have clHsub1:cl(H)×cl(H)⊆G×G by auto
  have Closure(H×H,ProductTopology(T,T))=cl(H)×cl(H) using cl_product
    topSpaceAssum group0.group0_3_L2 group0_valid_in_tgroup assms by auto
  then have f(Closure(H×H,ProductTopology(T,T)))=f(cl(H)×cl(H)) by auto
  with closure sub have clcl:f(cl(H)×cl(H))⊆cl(f(H×H)) by force
  from assms have fun:restrict(f,H×H):H×H→H unfolding IsAsubgroup_def
using
    group0.group_oper_fun unfolding group0_def by auto
  then have restrict(f,H×H)(H×H)=f(H×H) using restrict_image by auto
  moreover from fun have restrict(f,H×H)(H×H)⊆H using func1_1_L6(2)
by blast
  ultimately have f(H×H)⊆H by auto
  with sub1 have f(H×H)⊆Hf(H×H)⊆GH⊆G by auto
  then have cl(f(H×H))⊆cl(H) using top_closure_mono by auto
  with clcl have img:f(cl(H)×cl(H))⊆cl(H) by auto
  {
    fix x y assume x∈cl(H)y∈cl(H)
    then have ⟨x,y⟩∈cl(H)×cl(H) by auto moreover
```

```
    have f(cl(H)×cl(H))={ft. t∈cl(H)×cl(H)} using func_imagedef topgroup_f_binop

      clHsub1 by auto ultimately
    have f⟨x,y⟩∈f(cl(H)×cl(H)) by auto
    with img have f⟨x,y⟩∈cl(H) by auto
  }
  then have A1:cl(H){is closed under} f unfolding IsOpClosed_def by auto
  have two:two_top_spaces0(T,T,GroupInv(G,f)) unfolding two_top_spaces0_def
using
    topSpaceAssum Ggroup group0_2_T2 by auto
  from inv_cont have cont:IsContinuous(T,T,GroupInv(G,f)) by auto
  then have closed:∀D. D{is closed in}T ⟶ GroupInv(G,f)-D{is closed
in}T using two_top_spaces0.TopZF_2_1_L1
    two by auto
  then have closure:∀A∈Pow(⋃T). GroupInv(G,f)(cl(A))⊆cl(GroupInv(G,f)A)
using two_top_spaces0.Top_ZF_2_1_L2
    two by force
  with sub1 have Inv:GroupInv(G,f)(cl(H))⊆cl(GroupInv(G,f)H) by auto
moreover
  have GroupInv(H,restrict(f,H×H)):H→H using assms unfolding IsAsubgroup_def
using group0_2_T2 by auto then
  have GroupInv(H,restrict(f,H×H))H⊆H using func1_1_L6(2) by auto
  then have restrict(GroupInv(G,f),H)H⊆H using group0.group0_3_T1 assms
group0_valid_in_tgroup by auto
  then have sss:GroupInv(G,f)H⊆H using restrict_image by auto
  then have H⊆G GroupInv(G,f)H⊆G using sub1 by auto
  with sub1 sss have cl(GroupInv(G,f)H)⊆cl(H) using top_closure_mono
by auto ultimately
  have img:GroupInv(G,f)(cl(H))⊆cl(H) by auto
  {
    fix x assume x∈cl(H) moreover
    have GroupInv(G,f)(cl(H))={GroupInv(G,f)t. t∈cl(H)} using func_imagedef
Ggroup group0_2_T2
      clHG by force ultimately
    have GroupInv(G,f)x∈GroupInv(G,f)(cl(H)) by auto
    with img have GroupInv(G,f)x∈cl(H) by auto
  }
  then have A2:∀x∈cl(H). GroupInv(G,f)x∈cl(H) by auto
  from assms have H≠0 using group0.group0_3_L5 group0_valid_in_tgroup
by auto moreover
  have H⊆cl(H) using cl_contains_set sub1 by auto ultimately
  have cl(H)≠0 by auto
  with clHG A2 A1 show thesis using group0.group0_3_T3 group0_valid_in_tgroup
by auto
qed
```

The closure of a normal subgroup is normal.

```
theorem (in topgroup) normal_subg:
  assumes IsAnormalSubgroup(G,f,H)
```

```
  shows IsAnormalSubgroup(G,f,cl(H))
proof-
  have A:IsAsubgroup(cl(H),f) using closure_subgroup assms unfolding IsAnormalSubgroup_def
by auto
  have sub1:H⊆G using group0.group0_3_L2 group0_valid_in_tgroup assms
unfolding IsAnormalSubgroup_def by auto
  then have sub2:cl(H)⊆G using Top_3_L11(1) by auto
  {
    fix g assume g:g∈G
    then have cl1:cl(g+H)=g+cl(H) using trans_closure sub1 by auto
    have ss:g+cl(H)⊆G unfolding ltrans_def LeftTranslation_def by auto
    have g+H⊆G unfolding ltrans_def LeftTranslation_def by auto
    moreover from g have (-g)∈G using neg_in_tgroup by auto
    ultimately have cl2:cl((g+H)+(-g))=cl(g+H)+(-g) using trans_closure2
      by auto
    with cl1 have clcon:cl((g+H)+(-g))=(g+(cl(H)))+(-g) by auto
    {
      fix r assume r∈(g+H)+(-g)
      then obtain q where q:q∈g+H r=q+(-g) unfolding rtrans_def RightTranslation_def
        by force
      from q(1) obtain h where h∈H q=g+h unfolding ltrans_def LeftTranslation_def
by auto
      with q(2) have r=(g+h)+(-g) by auto
      with ‹h∈H› ‹g∈G› ‹(-g)∈G› have r∈H using assms unfolding IsAnormalSubgroup_def
        grinv_def grop_def by auto
    }
    then have (g+H)+(-g)⊆H by auto
    moreover then have (g+H)+(-g)⊆GH⊆G using sub1 by auto ultimately
    have cl((g+H)+(-g))⊆cl(H) using top_closure_mono by auto
    with clcon have (g+(cl(H)))+(-g)⊆cl(H) by auto moreover
    {
      fix b assume b∈{g+(d-g). d∈cl(H)}
      then obtain d where d:d∈cl(H) b=g+(d-g) by auto moreover
      then have d∈G using sub2 by auto
      then have g+d∈G using group0.group_op_closed[OF group0_valid_in_tgroup
‹g∈G›] by auto
      from d(2) have b:b=(g+d)-g using group0.group_oper_assoc[OF group0_valid_in_tgroup
‹g∈G› ‹d∈G›‹(-g)∈G›]
        unfolding grsub_def grop_def grinv_def by blast
      have (g+d)=LeftTranslation(G,f,g)d using group0.group0_5_L2(2)[OF
group0_valid_in_tgroup]
        ‹g∈G›‹d∈G› by auto
      with ‹d∈cl(H)› have g+d∈g+cl(H) unfolding ltrans_def using func_imagedef[OF
group0.group0_5_L1(2)[
        OF group0_valid_in_tgroup ‹g∈G›] sub2] by auto
      moreover from b have b=RightTranslation(G,f,-g)(g+d) using group0.group0_5_L2(1)[OF
group0_valid_in_tgroup]
        ‹(-g)∈G›‹g+d∈G› by auto
      ultimately have b∈(g+cl(H))+(-g) unfolding rtrans_def using func_imagedef[OF
```

```
group0.group0_5_L1(1)[
        OF group0_valid_in_tgroup ‹(-g)∈G›] ss] by force
    }
    ultimately have {g+(d-g). d∈cl(H)}⊆cl(H) by force
  }
  then show thesis using A group0.cont_conj_is_normal[OF group0_valid_in_tgroup,
of cl(H)]
    unfolding grsub_def grinv_def grop_def by auto
qed
```

Every open subgroup is also closed.

```
theorem (in topgroup) open_subgroup_closed:
  assumes IsAsubgroup(H,f) H∈T
  shows H{is closed in}T
proof-
  from assms(1) have sub:H⊆G using group0.group0_3_L2 group0_valid_in_tgroup
by force
  {
    fix t assume t∈G-H
    then have tnH:t∉H and tG:t∈G by auto
    from assms(1) have sub:H⊆G using group0.group0_3_L2 group0_valid_in_tgroup
by force
    from assms(1) have nSubG:0∈H using group0.group0_3_L5 group0_valid_in_tgroup
by auto
    from assms(2) tG have P:t+H∈T using open_tr_open(1) by auto
    from nSubG sub tG have tp:t∈t+H using group0_valid_in_tgroup group0.neut_trans_elem
      by auto
    {
      fix x assume x∈(t+H)∩H
      then obtain u where x=t+u u∈H x∈H unfolding ltrans_def LeftTranslation_def
by auto
      then have u∈Gx∈Gt∈G using sub tG by auto
      with ‹x=t+u› have x+(-u)=t using group0.group0_2_L18(1) group0_valid_in_tgroup
        unfolding grop_def grinv_def by auto
      from ‹u∈H› have (-u)∈H unfolding grinv_def using assms(1) group0.group0_3_T3A
group0_valid_in_tgroup
        by auto
      with ‹x∈H› have x+(-u)∈H unfolding grop_def using assms(1) group0.group0_3_L6
group0_valid_in_tgroup
        by auto
      with ‹x+(-u)=t› have False using tnH by auto
    }
    then have (t+H)∩H=0 by auto moreover
    have t+H⊆G unfolding ltrans_def LeftTranslation_def by auto ulti-
mately
    have (t+H)⊆G-H by auto
    with tp P have ∃V∈T. t∈V ∧ V⊆G-H unfolding Bex_def by auto
  }
  then have ∀t∈G-H. ∃V∈T. t∈V ∧ V⊆G-H by auto
```

```
  then have G-H∈T using open_neigh_open by auto
  then show thesis unfolding IsClosed_def using sub by auto
qed
```

Any subgroup with non-empty interior is open.

```
theorem (in topgroup) clopen_or_emptyInt:
  assumes IsAsubgroup(H,f) int(H)≠0
  shows H∈T
proof-
  from assms(1) have sub:H⊆G using group0.group0_3_L2 group0_valid_in_tgroup
by force
  {
    fix h assume h∈H
    have intsub:int(H)⊆H using Top_2_L1 by auto
    from assms(2) obtain u where u∈int(H) by auto
    with intsub have u∈H by auto
    then have (-u)∈H unfolding grinv_def using assms(1) group0.group0_3_T3A
group0_valid_in_tgroup
      by auto
    with ‹h∈H› have h-u∈H unfolding grop_def using assms(1) group0.group0_3_L6
group0_valid_in_tgroup
      by auto
    {
      fix t assume t∈(h-u)+(int(H))
      then obtain r where r∈int(H)t=(h-u)+r unfolding grsub_def grinv_def
grop_def
        ltrans_def LeftTranslation_def by auto
      then have r∈H using intsub by auto
      with ‹h-u∈H› have (h-u)+r∈H unfolding grop_def using assms(1) group0.group0_3_L6
group0_valid_in_tgroup
        by auto
      with ‹t=(h-u)+r› have t∈H by auto
    }
    then have ss:(h-u)+(int(H))⊆H by auto
    have P:(h-u)+(int(H))∈T using open_tr_open(1) ‹h-u∈H› Top_2_L2 sub
by blast
    from ‹h-u∈H›‹u∈H›‹h∈H› sub have (h-u)∈G u∈Gh∈G by auto
    have int(H)⊆G using sub intsub by auto moreover
    have LeftTranslation(G,f,(h-u))∈G→G using group0.group0_5_L1(2) group0_valid_in_tgroup
‹(h-u)∈G›
      by auto ultimately
    have LeftTranslation(G,f,(h-u))(int(H))={LeftTranslation(G,f,(h-u))r.
r∈int(H)}
      using func_imagedef by auto moreover
    from ‹(h-u)∈G› ‹u∈G› have LeftTranslation(G,f,(h-u))u=(h-u)+u us-
ing group0.group0_5_L2(2) group0_valid_in_tgroup
      by auto
    with ‹u∈int(H)› have (h-u)+u∈{LeftTranslation(G,f,(h-u))r. r∈int(H)}
by force ultimately
```

```
    have (h-u)+u∈(h-u)+(int(H)) unfolding ltrans_def by auto moreover
    have (h-u)+u=h using group0.inv_cancel_two(1) group0_valid_in_tgroup
      ‹u∈G›‹h∈G› by auto ultimately
    have h∈(h-u)+(int(H)) by auto
    with P ss have ∃V∈T. h∈V∧ V⊆H unfolding Bex_def by auto
  }
  then show thesis using open_neigh_open by auto
qed
```

In conclusion, a subgroup is either open or has empty interior.

```
corollary(in topgroup) emptyInterior_xor_op:
  assumes IsAsubgroup(H,f)
  shows (int(H)=0) Xor (H∈T)
  unfolding Xor_def using clopen_or_emptyInt assms Top_2_L3
  group0.group0_3_L5 group0_valid_in_tgroup by force
```

Then no connected topological groups has proper subgroups with non-empty interior.

```
corollary(in topgroup) connected_emptyInterior:
  assumes IsAsubgroup(H,f) T{is connected}
  shows (int(H)=0) Xor (H=G)
proof-
  have (int(H)=0) Xor (H∈T) using emptyInterior_xor_op assms(1) by auto
moreover
  {
    assume H∈T moreover
    then have H{is closed in}T using open_subgroup_closed assms(1) by
auto ultimately
    have H=0∨H=G using assms(2) unfolding IsConnected_def by auto
    then have H=G using group0.group0_3_L5 group0_valid_in_tgroup assms(1)
by auto
  } moreover
  have G∈T using topSpaceAssum unfolding IsATopology_def G_def by auto
  ultimately show thesis unfolding Xor_def by auto
qed
```

Every locally-compact subgroup of a T_0 group is closed.

```
theorem (in topgroup) loc_compact_T0_closed:
  assumes IsAsubgroup(H,f) (T{restricted to}H){is locally-compact} T{is
T0}
  shows H{is closed in}T
proof-
  from assms(1) have clsub:IsAsubgroup(cl(H),f) using closure_subgroup
by auto
  then have subcl:cl(H)⊆G using group0.group0_3_L2 group0_valid_in_tgroup
by force
  from assms(1) have sub:H⊆G using group0.group0_3_L2 group0_valid_in_tgroup
by force
```

```
  from assms(3) have T{is T_2} using T1_imp_T2 neu_closed_imp_T1 T0_imp_neu_closed
by auto
  then have (T{restricted to}H){is T_2} using T2_here sub by auto
  have tot:⋃(T{restricted to}H)=H using sub unfolding RestrictedTo_def
by auto
  with assms(2) have ∀x∈H. ∃A∈Pow(H). A {is compact in} (T{restricted
to}H) ∧ x ∈ Interior(A, (T{restricted to}H)) using
    topology0.locally_compact_exist_compact_neig[of T{restricted to}H]
Top_1_L4 unfolding topology0_def
    by auto
  then obtain K where K:K⊆H K{is compact in} (T{restricted to}H)0∈Interior(K,(T{restricted
to}H))
    using group0.group0_3_L5 group0_valid_in_tgroup assms(1) unfolding
gzero_def by force
  from K(1,2) have K{is compact in} T using compact_subspace_imp_compact
by auto
  with ‹T{is T_2}› have Kcl:K{is closed in}T using in_t2_compact_is_cl
by auto
  have Interior(K,(T{restricted to}H))∈(T{restricted to}H) using topology0.Top_2_L2
unfolding topology0_def
    using Top_1_L4 by auto
  then obtain U where U:U∈TInterior(K,(T{restricted to}H))=H∩U unfold-
ing RestrictedTo_def by auto
  then have H∩U⊆K using topology0.Top_2_L1[of T{restricted to}H] un-
folding topology0_def using Top_1_L4 by force
  moreover have U2:U⊆U∪K by auto
  have ksub:K⊆H using tot K(2) unfolding IsCompact_def by auto
  ultimately have int:H∩(U∪K)=K by auto
  from U(2) K(3) have 0∈U by auto
  with U(1) U2 have 0∈int(U ∪ K) using Top_2_L6 by auto
  then have U∪K∈𝒩_0 unfolding zerohoods_def using U(1) ksub sub by auto
  then obtain V where V:V⊆U∪K V∈𝒩_0 V+V⊆U∪K(- V) = V using exists_procls_zerohood[of
U∪K]
    by auto
  {
    fix h assume AS:h∈cl(H)
    with clsub have (-h)∈cl(H) using group0.group0_3_T3A group0_valid_in_tgroup
by auto moreover
    then have (-h)∈G using subcl by auto
    with V(2) have (-h)∈int((-h)+V) using elem_in_int_ltrans by auto
ultimately
    have (-h)∈(cl(H))∩(int((-h)+V)) by auto moreover
    have int((-h)+V)∈T using Top_2_L2 by auto moreover
    note sub ultimately
    have H∩(int((-h)+V))≠0 using cl_inter_neigh by auto moreover
    from ‹(-h)∈G› V(2) have int((-h)+V)=(-h)+int(V) unfolding zerohoods_def

      using ltrans_interior by force
    ultimately have H∩((-h)+int(V))≠0 by auto
```

```
    then obtain y where y:y∈H y∈(-h)+int(V) by blast
    then obtain v where v:v∈int(V) y=(-h)+v unfolding ltrans_def LeftTranslation_def
by auto
    with ‹(-h)∈G› V(2) y(1) sub have v∈G(-h)∈Gy∈G using Top_2_L1[of
V] unfolding zerohoods_def by auto
    with v(2) have (-(-h))+y=v using group0.group0_2_L18(2) group0_valid_in_tgroup
      unfolding grop_def grinv_def by auto moreover
    have h∈G using AS subcl by auto
    then have (-(-h))=h using group0.group_inv_of_inv group0_valid_in_tgroup
by auto ultimately
    have h+y=v by auto
    with v(1) have hyV:h+y∈int(V) by auto
    have y∈cl(H) using y(1) cl_contains_set sub by auto
    with AS have hycl:h+ y∈cl(H) using clsub group0.group0_3_L6 group0_valid_in_tgroup
by auto
    {
      fix W assume W:W∈Th+y∈W
      with hyV have h+y∈int(V)∩W by auto moreover
      from W(1) have int(V)∩W∈T using Top_2_L2 topSpaceAssum unfold-
ing IsATopology_def by auto moreover
      note hycl sub
      ultimately have (int(V)∩W)∩H≠0 using cl_inter_neigh[of Hint(V)∩Wh+y]
by auto
      then have V∩W∩H≠0 using Top_2_L1 by auto
      with V(1) have (U∪K)∩W∩H≠0 by auto
      then have (H∩(U∪K))∩W≠0 by auto
      with int have K∩W≠0 by auto
    }
    then have ∀W∈T. h+y∈W ⟶ K∩W≠0 by auto moreover
    have K⊆G h+y∈G using ksub sub hycl subcl by auto ultimately
    have h+y∈cl(K) using inter_neigh_cl[of Kh+y] unfolding G_def by force
    then have h+y∈K using Kcl Top_3_L8 ‹K⊆G› by auto
    with ksub have h+y∈H by auto
    moreover from y(1) have (-y)∈H using group0.group0_3_T3A assms(1)
group0_valid_in_tgroup
      by auto
    ultimately have (h+y)-y∈H unfolding grsub_def using group0.group0_3_L6
group0_valid_in_tgroup
      assms(1) by auto
    moreover
    have (-y)∈G using ‹(-y)∈H› sub by auto
    then have h+(y-y)=(h+y)-y using ‹y∈G›‹h∈G› group0.group_oper_assoc
      group0_valid_in_tgroup unfolding grsub_def by auto
    then have h+0=(h+y)-y using group0.group0_2_L6 group0_valid_in_tgroup
‹y∈G›
      unfolding grsub_def grinv_def grop_def gzero_def by auto
    then have h=(h+y)-y using group0.group0_2_L2 group0_valid_in_tgroup
      ‹h∈G› unfolding gzero_def by auto
    ultimately have h∈H by auto
```

```
  }
  then have cl(H)⊆H by auto
  then have H=cl(H) using cl_contains_set sub by auto
  then show thesis using Top_3_L8 sub by auto
qed
```

We can always consider a factor group which is T_2.

```
theorem(in topgroup) factor_haus:
  shows (T{quotient by}QuotientGroupRel(G,f,cl({0}))){is T2}
proof-
  let r=QuotientGroupRel(G,f,cl({0}))
  let f=QuotientGroupOp(G,f,cl({0}))
  let i=GroupInv(G//r,f)
  have IsAnormalSubgroup(G,f,{0}) using group0.trivial_normal_subgroup
Ggroup unfolding group0_def
    by auto
  then have normal:IsAnormalSubgroup(G,f,cl({0})) using normal_subg by
auto
  then have eq:equiv(⋃T,r) using group0.Group_ZF_2_4_L3[OF group0_valid_in_tgroup]
    unfolding IsAnormalSubgroup_def by auto
  then have tot:⋃(T{quotient by}r)=G//r using total_quo_equi by auto
  have neu:r{0}=TheNeutralElement(G//r,f) using Group_ZF_2_4_L5B[OF Ggroup
normal] by auto
  then have r{0}∈G//r using group0.group0_2_L2 Group_ZF_2_4_T1[OF Ggroup
normal] unfolding group0_def by auto
  then have sub1:{r{0}}⊆G//r by auto
  then have sub:{r{0}}⊆⋃(T{quotient by}r) using tot by auto
  have zG:0∈⋃T using group0.group0_2_L2[OF group0_valid_in_tgroup] by
auto
  from zG have cla:r{0}∈G//r unfolding quotient_def by auto
  let x=G//r-{r{0}}
  {
    fix s assume A:s∈⋃(G//r-{r{0}})
    then obtain U where s∈U U∈G//r-{r{0}} by auto
    then have U∈G//r U≠r{0} s∈U by auto
    then have U∈G//r s∈U s∉r{0} using cla quotient_disj[OF eq] by auto
    then have s∈⋃(G//r)-r{0} by auto
  }
  moreover
  {
    fix s assume A:s∈⋃(G//r)-r{0}
    then obtain U where s∈U U∈G//r s∉r{0} by auto
    then have s∈U U∈G//r-{r{0}} by auto
    then have s∈⋃(G//r-{r{0}}) by auto
  }
  ultimately have ⋃(G//r-{r{0}})=⋃(G//r)-r{0} by auto
  then have A:⋃(G//r-{r{0}})=G-r{0} using Union_quotient eq by auto
  {
    fix s assume A:s∈r{0}
```

```
    then have ⟨0,s⟩∈r by auto
    then have ⟨s,0⟩∈r using eq unfolding equiv_def sym_def by auto
    then have s∈cl({0}) using group0.Group_ZF_2_4_L5C[OF group0_valid_in_tgroup]
unfolding QuotientGroupRel_def by auto
  }
  moreover
  {
    fix s assume A:s∈cl({0})
    then have s∈G using Top_3_L11(1) zG by auto
    then have ⟨s,0⟩∈r using group0.Group_ZF_2_4_L5C[OF group0_valid_in_tgroup]
A by auto
    then have ⟨0,s⟩∈r using eq unfolding equiv_def sym_def by auto
    then have s∈r{0} by auto
  }
  ultimately have r{0}=cl({0}) by blast
  with A have ⋃(G//r-{r{0}})=G-cl({0}) by auto
  moreover have cl({0}){is closed in}T using cl_is_closed zG by auto
  ultimately have ⋃(G//r-{r{0}})∈T unfolding IsClosed_def by auto
  then have (G//r-{r{0}})∈{quotient by}r using quotient_equiv_rel eq
by auto
  then have (⋃(T{quotient by}r)-{r{0}})∈{quotient by}r using total_quo_equi[OF
eq] by auto
  moreover from sub1 have {r{0}}⊆(⋃(T{quotient by}r)) using total_quo_equi[OF
eq] by auto
  ultimately have {r{0}}{is closed in}(T{quotient by}r) unfolding IsClosed_def
by auto
  then have {TheNeutralElement(G//r,f)}{is closed in}(T{quotient by}r)
using neu by auto
  then have (T{quotient by}r){is T₁} using topgroup.neu_closed_imp_T1[OF
topGroupLocale[OF quotient_top_group[OF normal]]]
    total_quo_equi[OF eq] by auto
  then show thesis using topgroup.T1_imp_T2[OF topGroupLocale[OF quotient_top_group[OF
normal]]] by auto
qed


end
```

102 Metamath introduction

theory MMI_prelude **imports** Order_ZF_1

begin

Metamath's set.mm features a large (over 8000) collection of theorems proven in the ZFC set theory. This theory is part of an attempt to translate those theorems to Isar so that they are available for Isabelle/ZF users. A total of about 1200 assertions have been translated, 600 of that with proofs (the rest was proven automatically by Isabelle). The translation was done with the support of the mmisar tool, whose source is included in the IsarMathLib distributions prior to version 1.6.4. The translation tool was doing about 99 percent of work involved, with the rest mostly related to the difference between Isabelle/ZF and Metamath metalogics. Metamath uses Tarski-Megill metalogic that does not have a notion of bound variables (see `http://planetx.cc.vt.edu/AsteroidMeta/Distinctors_vs_binders` for details and discussion). The translation project is closed now as I decided that it was too boring and tedious even with the support of mmisar software. Also, the translated proofs are not as readable as native Isar proofs which goes against IsarMathLib philosophy.

102.1 Importing from Metamath - how is it done

We are interested in importing the theorems about complex numbers that start from the "recnt" theorem on. This is done mostly automatically by the mmisar tool that is included in the IsarMathLib distributions prior to version 1.6.4. The tool works as follows:

First it reads the list of (Metamath) names of theorems that are already imported to IsarMathlib ("known theorems") and the list of theorems that are intended to be imported in this session ("new theorems"). The new theorems are consecutive theorems about complex numbers as they appear in the Metamath database. Then mmisar creates a "Metamath script" that contains Metamath commands that open a log file and put the statements and proofs of the new theorems in that file in a readable format. The tool writes this script to a disk file and executes metamath with standard input redirected from that file. Then the log file is read and its contents converted to the Isar format. In Metamath, the proofs of theorems about complex numbers depend only on 28 axioms of complex numbers and some basic logic and set theory theorems. The tool finds which of these dependencies are not known yet and repeats the process of getting their statements from Metamath as with the new theorems. As a result of this process mmisar creates files new_theorems.thy, new_deps.thy and new_known_theorems.txt. The file new_theorems.thy contains the theorems (with proofs) imported from Metamath in this session. These theorems are added (by hand) to the current `MMI_Complex_ZF_x.thy` file. The file new_deps.thy contains the statements of new dependencies with generic proofs "by auto". These are added to the `MMI_logic_and_sets.thy`. Most of the dependencies can be proven automatically by Isabelle. However, some manual work has to be done for the dependencies that Isabelle can not prove by itself and to correct problems related to the fact that Metamath uses a metalogic based on distinct variable constraints (Tarski-Megill metalogic), rather than an explicit notion of free and bound variables.

The old list of known theorems is replaced by the new list and mmisar is ready to convert the next batch of new theorems. Of course this rarely works in practice without tweaking the mmisar source files every time a new batch is processed.

102.2 The context for Metamath theorems

We list the Metamth's axioms of complex numbers and define notation here.

The next definition is what Metamath $X \in V$ is translated to. I am not sure why it works, probably because Isabelle does a type inference and the "=" sign indicates that both sides are sets.

```
definition
  IsASet :: i⇒o (_ isASet [90] 90) where

  IsASet_def[simp]: X isASet ≡  X = X
```

The next locale sets up the context to which Metamath theorems about complex numbers are imported. It assumes the axioms of complex numbers and defines the notation used for complex numbers.

One of the problems with importing theorems from Metamath is that Metamath allows direct infix notation for binary operations so that the notation afb is allowed where f is a function (that is, a set of pairs). To my knowledge, Isar allows only notation `f⟨a,b⟩` with a possibility of defining a syntax say `a + b` to mean the same as `f⟨a,b⟩` (please correct me if I am wrong here). This is why we have two objects for addition: one called `caddset` that represents the binary function, and the second one called `ca` which defines the `a + b` notation for `caddset⟨a,b⟩`. The same applies to multiplication of real numbers.

Another difficulty is that Metamath allows to define sets with syntax $\{x|p\}$ where p is some formula that (usually) depends on x. Isabelle allows the set comprehension like this only as a subset of another set i.e. $\{x \in A.p(x)\}$. This forces us to have a sligtly different definition of (complex) natural numbers, requiring explicitly that natural numbers is a subset of reals. Because of that, the proofs of Metamath theorems that reference the definition directly can not be imported.

```
locale MMIsar0 =
  fixes real (ℝ)
  fixes complex (ℂ)
  fixes one (1)
  fixes zero (0)
  fixes iunit (i)
  fixes caddset (+)
  fixes cmulset (·)
  fixes lessrrel (<ℝ)
```

```
fixes ca (infixl + 69)
defines ca_def: a + b ≡ +⟨a,b⟩
fixes cm (infixl · 71)
defines cm_def: a · b ≡ ·⟨a,b⟩
fixes sub (infixl - 69)
defines sub_def: a - b ≡ ⋃ { x ∈ ℂ. b + x = a }
fixes cneg (-_ 95)
defines cneg_def: - a ≡ 0 - a
fixes cdiv (infixl / 70)
defines cdiv_def: a / b ≡ ⋃ { x ∈ ℂ. b · x = a }
fixes cpnf (+∞)
defines cpnf_def: +∞ ≡ ℂ
fixes cmnf (−∞)
defines cmnf_def: −∞ ≡ {ℂ}
fixes cxr (ℝ*)
defines cxr_def: ℝ* ≡ ℝ ∪ {+∞,−∞}
fixes cxn (ℕ)
defines cxn_def: ℕ ≡ ⋂ {N ∈ Pow(ℝ). 1 ∈ N ∧ (∀n. n∈N ⟶ n+1 ∈ N)}
fixes lessr (infix <ℝ 68)
defines lessr_def: a <ℝ b ≡ ⟨a,b⟩ ∈ <ℝ
fixes cltrrset (<)
defines cltrrset_def:
< ≡ (<ℝ ∩ ℝ×ℝ) ∪ {⟨−∞,+∞⟩} ∪
(ℝ×{+∞}) ∪ ({−∞}×ℝ )
fixes cltrr (infix < 68)
defines cltrr_def: a < b ≡ ⟨a,b⟩ ∈ <
fixes convcltrr (infix > 68)
defines convcltrr_def: a > b ≡ ⟨a,b⟩ ∈ converse(<)
fixes lsq (infix ≤ 68)
defines lsq_def: a ≤ b ≡ ¬ (b < a)
fixes two (2)
defines two_def: 2 ≡ 1+1
fixes three (3)
defines three_def: 3 ≡ 2+1
fixes four (4)
defines four_def: 4 ≡ 3+1
fixes five (5)
defines five_def: 5 ≡ 4+1
fixes six (6)
defines six_def: 6 ≡ 5+1
fixes seven (7)
defines seven_def: 7 ≡ 6+1
fixes eight (8)
defines eight_def: 8 ≡ 7+1
fixes nine (9)
defines nine_def: 9 ≡ 8+1

assumes MMI_pre_axlttri:
A ∈ ℝ ∧ B ∈ ℝ ⟶ (A <ℝ B ⟷ ¬(A=B ∨ B <ℝ A))
```

assumes MMI_pre_axlttrn:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \wedge C \in \mathbb{R} \longrightarrow ((A <_{\mathbb{R}} B \wedge B <_{\mathbb{R}} C) \longrightarrow A <_{\mathbb{R}} C)$
assumes MMI_pre_axltadd:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \wedge C \in \mathbb{R} \longrightarrow (A <_{\mathbb{R}} B \longrightarrow C+A <_{\mathbb{R}} C+B)$
assumes MMI_pre_axmulgt0:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow (\mathbf{0} <_{\mathbb{R}} A \wedge \mathbf{0} <_{\mathbb{R}} B \longrightarrow \mathbf{0} <_{\mathbb{R}} A \cdot B)$
assumes MMI_pre_axsup:
$A \subseteq \mathbb{R} \wedge A \neq 0 \wedge (\exists x \in \mathbb{R}. \forall y \in A. y <_{\mathbb{R}} x) \longrightarrow$
$(\exists x \in \mathbb{R}. (\forall y \in A. \neg(x <_{\mathbb{R}} y)) \wedge (\forall y \in \mathbb{R}. (y <_{\mathbb{R}} x \longrightarrow (\exists z \in A. y <_{\mathbb{R}} z))))$
assumes MMI_axresscn: $\mathbb{R} \subseteq \mathbb{C}$
assumes MMI_ax1ne0: $\mathbf{1} \neq \mathbf{0}$
assumes MMI_axcnex: $\mathbb{C}$ isASet
assumes MMI_axaddopr: $+ : (\mathbb{C} \times \mathbb{C}) \to \mathbb{C}$
assumes MMI_axmulopr: $\cdot : (\mathbb{C} \times \mathbb{C}) \to \mathbb{C}$
assumes MMI_axmulcom: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A \cdot B = B \cdot A$
assumes MMI_axaddcl: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A + B \in \mathbb{C}$
assumes MMI_axmulcl: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A \cdot B \in \mathbb{C}$
assumes MMI_axdistr:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A \cdot (B + C) = A \cdot B + A \cdot C$
assumes MMI_axaddcom: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A + B = B + A$
assumes MMI_axaddass:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A + B + C = A + (B + C)$
assumes MMI_axmulass:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A \cdot B \cdot C = A \cdot (B \cdot C)$
assumes MMI_ax1re: $\mathbf{1} \in \mathbb{R}$
assumes MMI_axi2m1: $i \cdot i + \mathbf{1} = \mathbf{0}$
assumes MMI_ax0id: $A \in \mathbb{C} \longrightarrow A + \mathbf{0} = A$
assumes MMI_axicn: $i \in \mathbb{C}$
assumes MMI_axnegex: $A \in \mathbb{C} \longrightarrow (\exists x \in \mathbb{C}. (A + x) = \mathbf{0})$
assumes MMI_axrecex: $A \in \mathbb{C} \wedge A \neq \mathbf{0} \longrightarrow (\exists x \in \mathbb{C}. A \cdot x = \mathbf{1})$
assumes MMI_ax1id: $A \in \mathbb{C} \longrightarrow A \cdot \mathbf{1} = A$
assumes MMI_axaddrcl: $A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow A + B \in \mathbb{R}$
assumes MMI_axmulrcl: $A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow A \cdot B \in \mathbb{R}$
assumes MMI_axrnegex: $A \in \mathbb{R} \longrightarrow (\exists x \in \mathbb{R}. A + x = \mathbf{0})$
assumes MMI_axrrecex: $A \in \mathbb{R} \wedge A \neq \mathbf{0} \longrightarrow (\exists x \in \mathbb{R}. A \cdot x = \mathbf{1})$

end

103 Logic and sets in Metamatah

theory MMI_logic_and_sets **imports** MMI_prelude

begin

103.1 Basic Metamath theorems

This section contains Metamath theorems that the more advanced theorems from `MMIsar.thy` depend on. Most of these theorems are proven automatically by Isabelle, some have to be proven by hand and some have to be modified to convert from Tarski-Megill metalogic used by Metamath to one based on explicit notion of free and bound variables.

```
lemma MMI_ax_mp: assumes φ and φ ⟶ ψ shows ψ
  using assms by auto

lemma MMI_sseli: assumes A1: A ⊆ B
   shows C ∈ A ⟶ C ∈ B
   using assms by auto

lemma MMI_sselii: assumes A1: A ⊆ B and
    A2: C ∈ A
   shows C ∈ B
   using assms by auto

lemma MMI_syl: assumes A1: φ ⟶ ps and
    A2: ps ⟶ ch
   shows φ ⟶ ch
   using assms by auto

lemma MMI_elimhyp: assumes A1: A = if ( φ , A , B ) ⟶ ( φ ⟷ ψ )
and
    A2: B = if ( φ , A , B ) ⟶ ( ch ⟷ ψ ) and
    A3: ch
   shows ψ
proof -
  { assume φ
    with A1 have ψ by simp }
  moreover
  { assume ¬φ
    with A2 A3 have ψ by simp }
  ultimately show ψ by auto
qed

lemma MMI_neeq1:
   shows A = B ⟶ ( A ≠ C ⟷ B ≠ C )
  by auto

lemma MMI_mp2: assumes A1: φ and
    A2: ψ and
    A3: φ ⟶ ( ψ ⟶ chi )
   shows chi
   using assms by auto
```

```
lemma MMI_xpex: assumes A1: A isASet and
    A2: B isASet
   shows ( A × B ) isASet
   using assms by auto

lemma MMI_fex:
   shows
  A ∈ C ⟶ ( F : A → B ⟶ F isASet )
  A isASet ⟶ ( F : A → B ⟶ F isASet )
  by auto

lemma MMI_3eqtr4d: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ C = A and
    A3: φ ⟶ D = B
   shows φ ⟶ C = D
   using assms by auto

lemma MMI_3coml: assumes A1: ( φ ∧ ψ ∧ chi ) ⟶ th
   shows ( ψ ∧ chi ∧ φ ) ⟶ th
   using assms by auto

lemma MMI_sylan: assumes A1: ( φ ∧ ψ ) ⟶ chi and
    A2: th ⟶ φ
   shows ( th ∧ ψ ) ⟶ chi
   using assms by auto

lemma MMI_3impa: assumes A1: ( ( φ ∧ ψ ) ∧ chi ) ⟶ th
   shows ( φ ∧ ψ ∧ chi ) ⟶ th
   using assms by auto

lemma MMI_3adant2: assumes A1: ( φ ∧ ψ ) ⟶ chi
   shows ( φ ∧ th ∧ ψ ) ⟶ chi
   using assms by auto

lemma MMI_3adant1: assumes A1: ( φ ∧ ψ ) ⟶ chi
   shows ( th ∧ φ ∧ ψ ) ⟶ chi
   using assms by auto

lemma (in MMIsar0) MMI_opreq12d: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ C = D
   shows
  φ ⟶ ( A + C ) = ( B + D )
  φ ⟶ ( A · C ) = ( B · D )
  φ ⟶ ( A - C ) = ( B - D )
  φ ⟶ ( A / C ) = ( B / D )
   using assms by auto

lemma MMI_mp2an: assumes A1: φ and
    A2: ψ and
```

```
    A3: ( φ ∧ ψ ) ⟶ chi
   shows chi
   using assms by auto

lemma MMI_mp3an: assumes A1: φ and
    A2: ψ and
    A3: ch and
    A4: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   shows ϑ
   using assms by auto

lemma MMI_eqeltrr: assumes A1: A = B and
    A2: A ∈ C
   shows B ∈ C
   using assms by auto

lemma MMI_eqtr: assumes A1: A = B and
    A2: B = C
   shows A = C
   using assms by auto



lemma MMI_impbi: assumes A1: φ ⟶ ψ and
    A2: ψ ⟶ φ
   shows φ ⟷ ψ
proof
  assume φ with A1 show ψ by simp
next
  assume ψ with A2 show φ by simp
qed

lemma MMI_mp3an3: assumes A1: ch and
    A2: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   shows ( φ ∧ ψ ) ⟶ ϑ
   using assms by auto

lemma MMI_eqeq12d: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ C = D
   shows φ ⟶ ( A = C ⟷ B = D )
   using assms by auto

lemma MMI_mpan2: assumes A1: ψ and
    A2: ( φ ∧ ψ ) ⟶ ch
   shows φ ⟶ ch
   using assms by auto

lemma (in MMIsar0) MMI_opreq2:
   shows
```

A = B ⟶ (C + A) = (C + B)
A = B ⟶ (C · A) = (C · B)
A = B ⟶ (C - A) = (C - B)
A = B ⟶ (C / A) = (C / B)
by auto

lemma MMI_syl5bir: **assumes** A1: φ ⟶ (ψ ⟷ ch) **and**
A2: ϑ ⟶ ch
shows φ ⟶ (ϑ ⟶ ψ)
using assms **by** auto

lemma MMI_adantr: **assumes** A1: φ ⟶ ψ
shows (φ ∧ ch) ⟶ ψ
using assms **by** auto

lemma MMI_mpan: **assumes** A1: φ **and**
A2: (φ ∧ ψ) ⟶ ch
shows ψ ⟶ ch
using assms **by** auto

lemma MMI_eqeq1d: **assumes** A1: φ ⟶ A = B
shows φ ⟶ (A = C ⟷ B = C)
using assms **by** auto

lemma (**in** MMIsar0) MMI_opreq1:
shows
A = B ⟶ (A · C) = (B · C)
A = B ⟶ (A + C) = (B + C)
A = B ⟶ (A - C) = (B - C)
A = B ⟶ (A / C) = (B / C)
by auto

lemma MMI_syl6eq: **assumes** A1: φ ⟶ A = B **and**
A2: B = C
shows φ ⟶ A = C
using assms **by** auto

lemma MMI_syl6bi: **assumes** A1: φ ⟶ (ψ ⟷ ch) **and**
A2: ch ⟶ ϑ
shows φ ⟶ (ψ ⟶ ϑ)
using assms **by** auto

lemma MMI_imp: **assumes** A1: φ ⟶ (ψ ⟶ ch)
shows (φ ∧ ψ) ⟶ ch
using assms **by** auto

lemma MMI_sylibd: **assumes** A1: φ ⟶ (ψ ⟶ ch) **and**
A2: φ ⟶ (ch ⟷ ϑ)
shows φ ⟶ (ψ ⟶ ϑ)

```
  using assms by auto

lemma MMI_ex: assumes A1: ( φ ∧ ψ ) ⟶ ch
  shows φ ⟶ ( ψ ⟶ ch )
  using assms by auto

lemma MMI_r19_23aiv: assumes A1: ∀x.  (x ∈ A  ⟶ (φ(x) ⟶ ψ ))
  shows ( ∃ x ∈ A . φ(x) ) ⟶ ψ
 using assms by auto

lemma MMI_bitr: assumes A1: φ ⟷ ψ and
   A2: ψ ⟷ ch
  shows φ ⟷ ch
  using assms by auto

lemma MMI_eqeq12i: assumes A1: A = B and
   A2: C = D
  shows A = C ⟷ B = D
  using assms by auto

lemma MMI_dedth3h:
 assumes A1: A = if ( φ , A , D ) ⟶ ( ϑ ⟷ ta ) and
   A2: B = if ( ψ , B , R ) ⟶ ( ta ⟷ et ) and
   A3: C = if ( ch , C , S ) ⟶ ( et ⟷ ze ) and
   A4: ze
  shows ( φ ∧ ψ ∧ ch ) ⟶ ϑ
  using assms by auto

lemma MMI_bibi1d: assumes A1: φ ⟶ ( ψ ⟷ ch )
  shows φ ⟶ ( ( ψ ⟷ ϑ ) ⟷ ( ch ⟷ ϑ ) )
  using assms by auto

lemma MMI_eqeq1:
  shows A = B ⟶ ( A = C ⟷ B = C )
 by auto

lemma MMI_bibi12d: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
   A2: φ ⟶ ( ϑ ⟷ ta )
  shows φ ⟶ ( ( ψ ⟷ ϑ ) ⟷ ( ch ⟷ ta ) )
  using assms by auto

lemma MMI_eqeq2d: assumes A1: φ ⟶ A = B
  shows φ ⟶ ( C = A ⟷ C = B )
  using assms by auto

lemma MMI_eqeq2:
  shows A = B ⟶ ( C = A ⟷ C = B )
 by auto
```

lemma MMI_elimel: **assumes** A1: B $\in$ C
shows if (A $\in$ C , A , B) $\in$ C
using assms **by** auto

lemma MMI_3adant3: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch
shows (φ $\wedge$ ψ $\wedge$ ϑ) $\longrightarrow$ ch
using assms **by** auto

lemma MMI_bitr3d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ϑ)
shows φ $\longrightarrow$ (ch $\longleftrightarrow$ ϑ)
using assms **by** auto

lemma MMI_3eqtr3d: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: φ $\longrightarrow$ A = C **and**
A3: φ $\longrightarrow$ B = D
shows φ $\longrightarrow$ C = D
using assms **by** auto

lemma (**in** MMIsar0) MMI_opreq1d: **assumes** A1: φ $\longrightarrow$ A = B
shows
φ $\longrightarrow$ (A + C) = (B + C)
φ $\longrightarrow$ (A - C) = (B - C)
φ $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ C)
φ $\longrightarrow$ (A / C) = (B / C)
using assms **by** auto

lemma MMI_3com12: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows (ψ $\wedge$ φ $\wedge$ ch) $\longrightarrow$ ϑ
using assms **by** auto

lemma (**in** MMIsar0) MMI_opreq2d: **assumes** A1: φ $\longrightarrow$ A = B
shows
φ $\longrightarrow$ (C + A) = (C + B)
φ $\longrightarrow$ (C - A) = (C - B)
φ $\longrightarrow$ (C $\cdot$ A) = (C $\cdot$ B)
φ $\longrightarrow$ (C / A) = (C / B)
using assms **by** auto

lemma MMI_3com23: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows (φ $\wedge$ ch $\wedge$ ψ) $\longrightarrow$ ϑ
using assms **by** auto

lemma MMI_3expa: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows ((φ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ ϑ
using assms **by** auto

```
lemma MMI_adantrr: assumes A1: ( φ ∧ ψ ) ⟶ ch
   shows ( φ ∧ ( ψ ∧ ϑ ) ) ⟶ ch
   using assms by auto

lemma MMI_3expb: assumes A1: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   shows ( φ ∧ ( ψ ∧ ch ) ) ⟶ ϑ
   using assms by auto

lemma MMI_an4s: assumes A1: ( ( φ ∧ ψ ) ∧ ( ch ∧ ϑ ) ) ⟶ τ
   shows ( ( φ ∧ ch ) ∧ ( ψ ∧ ϑ ) ) ⟶ τ
   using assms by auto

lemma MMI_eqtrd: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ B = C
   shows φ ⟶ A = C
   using assms by auto

lemma MMI_ad2ant2l: assumes A1: ( φ ∧ ψ ) ⟶ ch
   shows ( ( ϑ ∧ φ ) ∧ ( τ ∧ ψ ) ) ⟶ ch
   using assms by auto

lemma MMI_pm3_2i: assumes A1: φ and
    A2: ψ
   shows φ ∧ ψ
   using assms by auto

lemma (in MMIsar0) MMI_opreq2i: assumes A1: A = B
   shows
  ( C + A ) = ( C + B )
  ( C - A ) = ( C - B )
  ( C · A ) = ( C · B )
   using assms by auto



lemma MMI_mpbir2an: assumes A1: φ ⟷ ( ψ ∧ ch ) and
    A2: ψ and
    A3: ch
   shows φ
   using assms by auto

lemma MMI_reu4: assumes A1: ∀x y. x = y ⟶ ( φ(x) ⟷ ψ(y) )
   shows ( ∃! x . x ∈ A ∧ φ(x) ) ⟷
  ( ( ∃ x ∈ A . φ(x) ) ∧ ( ∀ x ∈ A . ∀ y ∈ A .
  ( ( φ(x) ∧ ψ(y) ) ⟶ x = y ) ) )
   using assms by auto

lemma MMI_risset:
   shows A ∈ B ⟷ ( ∃ x ∈ B . x = A )
```

```
  by auto

lemma MMI_sylib: assumes A1: φ ⟶ ψ and
    A2: ψ ⟷ ch
   shows φ ⟶ ch
   using assms by auto

lemma MMI_mp3an13: assumes A1: φ and
    A2: ch and
    A3: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   shows ψ ⟶ ϑ
   using assms by auto

lemma MMI_eqcomd: assumes A1: φ ⟶ A = B
   shows φ ⟶ B = A
   using assms by auto

lemma MMI_sylan9eqr: assumes A1: φ ⟶ A = B and
    A2: ψ ⟶ B = C
   shows ( ψ ∧ φ ) ⟶ A = C
   using assms by auto

lemma MMI_exp32: assumes A1: ( φ ∧ ( ψ ∧ ch ) ) ⟶ ϑ
   shows φ ⟶ ( ψ ⟶ ( ch ⟶ ϑ ) )
   using assms by auto

lemma MMI_impcom: assumes A1: φ ⟶ ( ψ ⟶ ch )
   shows ( ψ ∧ φ ) ⟶ ch
   using assms by auto

lemma MMI_a1d: assumes A1: φ ⟶ ψ
   shows φ ⟶ ( ch ⟶ ψ )
   using assms by auto

lemma MMI_r19_21aiv: assumes A1: ∀x. φ ⟶ ( x ∈ A ⟶ ψ(x) )
   shows φ ⟶ ( ∀ x ∈ A . ψ(x) )
   using assms by auto

lemma MMI_r19_22:
   shows ( ∀ x ∈ A . ( φ(x) ⟶ ψ(x) ) ) ⟶
  ( ( ∃ x ∈ A . φ(x) ) ⟶ ( ∃ x ∈ A . ψ(x) ) )
  by auto

lemma MMI_syl6: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: ch ⟶ ϑ
   shows φ ⟶ ( ψ ⟶ ϑ )
   using assms by auto

lemma MMI_mpid: assumes A1: φ ⟶ ch and
```

```
    A2: φ ⟶ ( ψ ⟶ ( ch ⟶ ϑ ) )
   shows φ ⟶ ( ψ ⟶ ϑ )
   using assms by auto

lemma MMI_eqtr3t:
   shows ( A = C ∧ B = C ) ⟶ A = B
  by auto

lemma MMI_syl5bi: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: ϑ ⟶ ψ
   shows φ ⟶ ( ϑ ⟶ ch )
   using assms by auto

lemma MMI_mp3an1: assumes A1: φ and
    A2: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   shows ( ψ ∧ ch ) ⟶ ϑ
   using assms by auto

lemma MMI_rgen2: assumes A1: ∀x y. ( x ∈ A ∧ y ∈ A ) ⟶ φ(x,y)
   shows ∀ x ∈ A . ∀ y ∈ A . φ(x,y)
   using assms by auto



lemma MMI_ax_17: shows φ ⟶ (∀x. φ) by simp


lemma MMI_3eqtr4g: assumes A1: φ ⟶ A = B and
    A2: C = A and
    A3: D = B
   shows φ ⟶ C = D
   using assms by auto



lemma MMI_3imtr4: assumes A1: φ ⟶ ψ and
    A2: ch ⟷ φ and
    A3: ϑ ⟷ ψ
   shows ch ⟶ ϑ
   using assms by auto



lemma MMI_eleq2i: assumes A1: A = B
   shows C ∈ A ⟷ C ∈ B
   using assms by auto

lemma MMI_albii: assumes A1: φ ⟷ ψ
   shows ( ∀ x . φ ) ⟷ ( ∀ x . ψ )
```

```
   using assms by auto


lemma MMI_reucl:
   shows ( ∃! x . x ∈ A ∧ φ(x) ) ⟶ ⋃ { x ∈ A . φ(x) } ∈ A
proof
  assume A1: ∃! x . x ∈ A ∧ φ(x)
  then obtain a where I: a∈A  and φ(a) by auto
  with A1 have { x ∈ A . φ(x) } = {a} by blast
  with I show  ⋃ { x ∈ A . φ(x) } ∈ A by simp
qed

lemma MMI_dedth2h: assumes A1: A = if ( φ , A , C ) ⟶ ( ch ⟷ ϑ
) and
    A2: B = if ( ψ , B , D ) ⟶ ( ϑ ⟷ τ ) and
    A3: τ
   shows ( φ ∧ ψ ) ⟶ ch
   using assms by auto

lemma MMI_eleq1d: assumes A1: φ ⟶ A = B
   shows φ ⟶ ( A ∈ C ⟷ B ∈ C )
   using assms by auto

lemma MMI_syl5eqel: assumes A1: φ ⟶ A ∈ B and
    A2: C = A
   shows φ ⟶ C ∈ B
   using assms by auto



lemma IML_eeuni: assumes A1: x ∈ A and A2: ∃! t . t ∈ A ∧ φ(t)
  shows φ(x) ⟷ ⋃ { x ∈ A . φ(x) } = x
proof
  assume φ(x)
  with A1 A2 show ⋃ { x ∈ A . φ(x) } = x by auto
next assume A3: ⋃ { x ∈ A . φ(x) } = x
  from A2 obtain y where y∈A and I: φ(y) by auto
  with A2 A3 have x = y by auto
  with I show φ(x) by simp
qed

lemma MMI_reuuni1:
   shows ( x ∈ A ∧ ( ∃! x . x ∈ A ∧ φ(x) ) ) ⟶
  ( φ(x) ⟷ ⋃ { x ∈ A . φ(x) } = x )
  using IML_eeuni by simp

lemma MMI_eqeq1i: assumes A1: A = B
   shows A = C ⟷ B = C
   using assms by auto
```

lemma MMI_syl6rbbr: **assumes** A1: $\forall$x. φ(x) $\longrightarrow$ (ψ(x) $\longleftrightarrow$ ch(x)) **and**
A2: $\forall$x. ϑ(x) $\longleftrightarrow$ ch(x)
shows $\forall$ x. φ(x) $\longrightarrow$ (ϑ(x) $\longleftrightarrow$ ψ(x))
using assms **by** auto

lemma MMI_syl6rbbrA: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: ϑ $\longleftrightarrow$ ch
shows φ $\longrightarrow$ (ϑ $\longleftrightarrow$ ψ)
using assms **by** auto

lemma MMI_vtoclga: **assumes** A1: $\forall$x. x = A $\longrightarrow$ (φ(x) $\longleftrightarrow$ ψ) **and**
A2: $\forall$x. x $\in$ B $\longrightarrow$ φ(x)
shows A $\in$ B $\longrightarrow$ ψ
using assms **by** auto

lemma MMI_3bitr4: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
A2: ch $\longleftrightarrow$ φ **and**
A3: ϑ $\longleftrightarrow$ ψ
shows ch $\longleftrightarrow$ ϑ
using assms **by** auto

lemma MMI_mpbii: **assumes** Amin: ψ **and**
Amaj: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ch
using assms **by** auto

lemma MMI_eqid:
shows A = A
by auto

lemma MMI_pm3_27:
shows (φ $\wedge$ ψ) $\longrightarrow$ ψ
by auto

lemma MMI_pm3_26:
shows (φ $\wedge$ ψ) $\longrightarrow$ φ
by auto

lemma MMI_ancoms: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch
shows (ψ $\wedge$ φ) $\longrightarrow$ ch
using assms **by** auto

lemma MMI_syl3anc: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ **and**
A2: τ $\longrightarrow$ φ **and**

```
    A3: τ ⟶ ψ and
    A4: τ ⟶ ch
   shows τ ⟶ ϑ
   using assms by auto

lemma MMI_syl5eq: assumes A1: φ ⟶ A = B and
    A2: C = A
   shows φ ⟶ C = B
   using assms by auto

lemma MMI_eqcomi: assumes A1: A = B
   shows B = A
   using assms by auto

lemma MMI_3eqtr: assumes A1: A = B and
    A2: B = C and
    A3: C = D
   shows A = D
   using assms by auto

lemma MMI_mpbir: assumes Amin: ψ and
    Amaj: φ ⟷ ψ
   shows φ
   using assms by auto

lemma MMI_syl3an3: assumes A1: ( φ ∧ ψ ∧ ch ) ⟶ ϑ and
    A2: τ ⟶ ch
   shows ( φ ∧ ψ ∧ τ ) ⟶ ϑ
   using assms by auto

lemma MMI_3eqtrd: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ B = C and
    A3: φ ⟶ C = D
   shows φ ⟶ A = D
   using assms by auto

lemma MMI_syl5: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: ϑ ⟶ ψ
   shows φ ⟶ ( ϑ ⟶ ch )
   using assms by auto

lemma MMI_exp3a: assumes A1: φ ⟶ ( ( ψ ∧ ch ) ⟶ ϑ )
   shows φ ⟶ ( ψ ⟶ ( ch ⟶ ϑ ) )
   using assms by auto

lemma MMI_com12: assumes A1: φ ⟶ ( ψ ⟶ ch )
   shows ψ ⟶ ( φ ⟶ ch )
   using assms by auto
```

```
lemma MMI_3imp: assumes A1: φ ⟶ ( ψ ⟶ ( ch ⟶ ϑ ) )
   shows ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   using assms by auto



lemma MMI_3eqtr3: assumes A1: A = B and
    A2: A = C and
    A3: B = D
   shows C = D
   using assms by auto

lemma (in MMIsar0) MMI_opreq1i: assumes A1: A = B
   shows
  ( A + C ) = ( B + C )
  ( A - C ) = ( B - C )
  ( A / C ) = ( B / C )
  ( A · C ) = ( B · C )
   using assms by auto

lemma MMI_eqtr3: assumes A1: A = B and
    A2: A = C
   shows B = C
   using assms by auto

lemma MMI_dedth: assumes A1: A = if ( φ , A , B ) ⟶ ( ψ ⟷ ch )
and
    A2: ch
   shows φ ⟶ ψ
   using assms by auto

lemma MMI_id:
   shows φ ⟶ φ
  by auto

lemma MMI_eqtr3d: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ A = C
   shows φ ⟶ B = C
   using assms by auto

lemma MMI_sylan2: assumes A1: ( φ ∧ ψ ) ⟶ ch and
    A2: ϑ ⟶ ψ
   shows ( φ ∧ ϑ ) ⟶ ch
   using assms by auto

lemma MMI_adantl: assumes A1: φ ⟶ ψ
   shows ( ch ∧ φ ) ⟶ ψ
   using assms by auto
```

```
lemma (in MMIsar0) MMI_opreq12:
   shows
  ( A = B ∧ C = D ) ⟶ ( A + C ) = ( B + D )
  ( A = B ∧ C = D ) ⟶ ( A - C ) = ( B - D )
  ( A = B ∧ C = D ) ⟶ ( A · C ) = ( B · D )
  ( A = B ∧ C = D ) ⟶ ( A / C ) = ( B / D )
  by auto

lemma MMI_anidms: assumes A1: ( φ ∧ φ ) ⟶ ψ
   shows φ ⟶ ψ
   using assms by auto



lemma MMI_anabsan2: assumes A1: ( φ ∧ ( ψ ∧ ψ ) ) ⟶ ch
   shows ( φ ∧ ψ ) ⟶ ch
   using assms by auto

lemma MMI_3simp2:
   shows ( φ ∧ ψ ∧ ch ) ⟶ ψ
  by auto

lemma MMI_3simp3:
   shows ( φ ∧ ψ ∧ ch ) ⟶ ch
  by auto

lemma MMI_sylbir: assumes A1: ψ ⟷ φ and
    A2: ψ ⟶ ch
   shows φ ⟶ ch
   using assms by auto

lemma MMI_3eqtr3g: assumes A1: φ ⟶ A = B and
    A2: A = C and
    A3: B = D
   shows φ ⟶ C = D
   using assms by auto

lemma MMI_3bitr: assumes A1: φ ⟷ ψ and
    A2: ψ ⟷ ch and
    A3: ch ⟷ ϑ
   shows φ ⟷ ϑ
   using assms by auto



lemma MMI_3bitr3: assumes A1: φ ⟷ ψ and
    A2: φ ⟷ ch and
    A3: ψ ⟷ ϑ
   shows ch ⟷ ϑ
```

```
  using assms by auto

lemma MMI_eqcom:
  shows A = B ⟷ B = A
 by auto

lemma MMI_syl6bb: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
   A2: ch ⟷ ϑ
  shows φ ⟶ ( ψ ⟷ ϑ )
  using assms by auto

lemma MMI_3bitr3d: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
   A2: φ ⟶ ( ψ ⟷ ϑ ) and
   A3: φ ⟶ ( ch ⟷ τ )
  shows φ ⟶ ( ϑ ⟷ τ )
  using assms by auto

lemma MMI_syl3an2: assumes A1: ( φ ∧ ψ ∧ ch ) ⟶ ϑ and
   A2: τ ⟶ ψ
  shows ( φ ∧ τ ∧ ch ) ⟶ ϑ
  using assms by auto



lemma MMI_df_rex:
  shows ( ∃ x ∈ A . φ(x) ) ⟷ ( ∃ x . ( x ∈ A ∧ φ(x) ) )
 by auto

lemma MMI_mpbi: assumes Amin: φ and
   Amaj: φ ⟷ ψ
  shows ψ
  using assms by auto

lemma MMI_mp3an12: assumes A1: φ and
   A2: ψ and
   A3: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
  shows ch ⟶ ϑ
  using assms by auto

lemma MMI_syl5bb: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
   A2: ϑ ⟷ ψ
  shows φ ⟶ ( ϑ ⟷ ch )
  using assms by auto

lemma MMI_eleq1a:
  shows A ∈ B ⟶ ( C = A ⟶ C ∈ B )
 by auto

lemma MMI_sylbird: assumes A1: φ ⟶ ( ch ⟷ ψ ) and
```

```
    A2: φ ⟶ ( ch ⟶ ϑ )
   shows φ ⟶ ( ψ ⟶ ϑ )
   using assms by auto

lemma MMI_19_23aiv: assumes A1: ∀x. φ(x) ⟶ ψ
   shows ( ∃ x . φ(x) ) ⟶ ψ
   using assms by auto

lemma MMI_eqeltrrd: assumes A1: φ ⟶ A = B and
    A2: φ ⟶ A ∈ C
   shows φ ⟶ B ∈ C
   using assms by auto

lemma MMI_syl2an: assumes A1: ( φ ∧ ψ ) ⟶ ch and
    A2: ϑ ⟶ φ and
    A3: τ ⟶ ψ
   shows ( ϑ ∧ τ ) ⟶ ch
   using assms by auto
```

```
lemma MMI_adantrl: assumes A1: ( φ ∧ ψ ) ⟶ ch
   shows ( φ ∧ ( ϑ ∧ ψ ) ) ⟶ ch
   using assms by auto

lemma MMI_ad2ant2r: assumes A1: ( φ ∧ ψ ) ⟶ ch
   shows ( ( φ ∧ ϑ ) ∧ ( ψ ∧ τ ) ) ⟶ ch
   using assms by auto

lemma MMI_adantll: assumes A1: ( φ ∧ ψ ) ⟶ ch
   shows ( ( ϑ ∧ φ ) ∧ ψ ) ⟶ ch
   using assms by auto

lemma MMI_anandirs: assumes A1: ( ( φ ∧ ch ) ∧ ( ψ ∧ ch ) ) ⟶ τ
   shows ( ( φ ∧ ψ ) ∧ ch ) ⟶ τ
   using assms by auto

lemma MMI_adantlr: assumes A1: ( φ ∧ ψ ) ⟶ ch
   shows ( ( φ ∧ ϑ ) ∧ ψ ) ⟶ ch
   using assms by auto

lemma MMI_an42s: assumes A1: ( ( φ ∧ ψ ) ∧ ( ch ∧ ϑ ) ) ⟶ τ
   shows ( ( φ ∧ ch ) ∧ ( ϑ ∧ ψ ) ) ⟶ τ
   using assms by auto
```

```
lemma MMI_mp3an2: assumes A1: ψ and
    A2: ( φ ∧ ψ ∧ ch ) ⟶ ϑ
```

```
  shows ( φ ∧ ch ) ⟶ ϑ
  using assms by auto



lemma MMI_3simp1:
  shows ( φ ∧ ψ ∧ ch ) ⟶ φ
 by auto

lemma MMI_3impb: assumes A1: ( φ ∧ ( ψ ∧ ch ) ) ⟶ ϑ
  shows ( φ ∧ ψ ∧ ch ) ⟶ ϑ
  using assms by auto

lemma MMI_mpbird: assumes Amin: φ ⟶ ch and
   Amaj: φ ⟶ ( ψ ⟷ ch )
  shows φ ⟶ ψ
  using assms by auto

lemma (in MMIsar0) MMI_opreq12i: assumes A1: A = B and
 A2: C = D
 shows
 ( A + C ) = ( B + D )
 ( A · C ) = ( B · D )
 ( A - C ) = ( B - D )
 using assms by auto

lemma MMI_3eqtr4: assumes A1: A = B and
 A2: C = A and
 A3: D = B
 shows C = D
 using assms by auto



lemma MMI_eqtr4d: assumes A1: φ ⟶ A = B and
   A2: φ ⟶ C = B
  shows φ ⟶ A = C
  using assms by auto



lemma MMI_3eqtr3rd: assumes A1: φ ⟶ A = B and
   A2: φ ⟶ A = C and
   A3: φ ⟶ B = D
  shows φ ⟶ D = C
  using assms by auto

lemma MMI_sylanc: assumes A1: ( φ ∧ ψ ) ⟶ ch and
   A2: ϑ ⟶ φ and
```

A3: $\vartheta \longrightarrow \psi$
shows $\vartheta \longrightarrow$ ch
using assms **by** auto

lemma MMI_anim12i: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch $\longrightarrow \vartheta$
shows ($\varphi \wedge$ ch) $\longrightarrow$ ($\psi \wedge \vartheta$)
using assms **by** auto

lemma (**in** MMIsar0) MMI_opreqan12d: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\psi \longrightarrow$ C = D
shows
($\varphi \wedge \psi$) $\longrightarrow$ (A + C) = (B + D)
($\varphi \wedge \psi$) $\longrightarrow$ (A - C) = (B - D)
($\varphi \wedge \psi$) $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ D)
using assms **by** auto

lemma MMI_sylanr2: **assumes** A1: ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow$ ch
shows ($\varphi \wedge$ ($\psi \wedge \tau$)) $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_sylanl2: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \psi$
shows (($\varphi \wedge \tau$) $\wedge$ ch) $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_ancom2s: **assumes** A1: ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$
shows ($\varphi \wedge$ (ch $\wedge \psi$)) $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_anandis: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ($\varphi \wedge$ ch)) $\longrightarrow \tau$
shows ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \tau$
using assms **by** auto

lemma MMI_sylan9eq: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\psi \longrightarrow$ B = C
shows ($\varphi \wedge \psi$) $\longrightarrow$ A = C
using assms **by** auto

lemma MMI_keephyp: **assumes** A1: A = if (φ , A , B) $\longrightarrow$ ($\psi \longleftrightarrow \vartheta$) **and**
A2: B = if (φ , A , B) $\longrightarrow$ (ch $\longleftrightarrow \vartheta$) **and**
A3: ψ **and**
A4: ch

```
   shows ϑ
proof -
  { assume φ
    with A1 A3 have ϑ by simp }
  moreover
  { assume ¬φ
    with A2 A4 have ϑ by simp }
  ultimately show ϑ by auto
qed

lemma MMI_eleq1:
   shows A = B ⟶ ( A ∈ C ⟷ B ∈ C )
  by auto

lemma MMI_pm4_2i:
   shows φ ⟶ ( ψ ⟷ ψ )
  by auto

lemma MMI_3anbi123d: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: φ ⟶ ( ϑ ⟷ τ ) and
    A3: φ ⟶ ( η ⟷ ζ )
   shows φ ⟶ ( ( ψ ∧ ϑ ∧ η ) ⟷ ( ch ∧ τ ∧ ζ ) )
   using assms by auto

lemma MMI_imbi12d: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: φ ⟶ ( ϑ ⟷ τ )
   shows φ ⟶ ( ( ψ ⟶ ϑ ) ⟷ ( ch ⟶ τ ) )
   using assms by auto

lemma MMI_bitrd: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: φ ⟶ ( ch ⟷ ϑ )
   shows φ ⟶ ( ψ ⟷ ϑ )
   using assms by auto

lemma MMI_df_ne:
   shows ( A ≠ B ⟷ ¬ ( A = B ) )
  by auto

lemma MMI_3pm3_2i: assumes A1: φ and
    A2: ψ and
    A3: ch
   shows φ ∧ ψ ∧ ch
   using assms by auto

lemma MMI_eqeq2i: assumes A1: A = B
   shows C = A ⟷ C = B
   using assms by auto

lemma MMI_syl5bbr: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
```

```
    A2: ψ ⟷ ϑ
   shows φ ⟶ ( ϑ ⟷ ch )
   using assms by auto

lemma MMI_biimpd: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ψ ⟶ ch )
   using assms by auto

lemma MMI_orrd: assumes A1: φ ⟶ ( ¬ ( ψ ) ⟶ ch )
   shows φ ⟶ ( ψ ∨ ch )
   using assms by auto

lemma MMI_jaoi: assumes A1: φ ⟶ ψ and
    A2: ch ⟶ ψ
   shows ( φ ∨ ch ) ⟶ ψ
   using assms by auto

lemma MMI_oridm:
   shows ( φ ∨ φ ) ⟷ φ
  by auto

lemma MMI_orbi1d: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ( ψ ∨ ϑ ) ⟷ ( ch ∨ ϑ ) )
   using assms by auto

lemma MMI_orbi2d: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ( ϑ ∨ ψ ) ⟷ ( ϑ ∨ ch ) )
   using assms by auto



lemma MMI_3bitr4g: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: ϑ ⟷ ψ and
    A3: τ ⟷ ch
   shows φ ⟶ ( ϑ ⟷ τ )
   using assms by auto

lemma MMI_negbid: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ¬ ( ψ ) ⟷ ¬ ( ch ) )
   using assms by auto

lemma MMI_ioran:
   shows ¬ ( ( φ ∨ ψ ) ) ⟷
 ( ¬ ( φ ) ∧ ¬ ( ψ ) )
  by auto

lemma MMI_syl6rbb: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: ch ⟷ ϑ
   shows φ ⟶ ( ϑ ⟷ ψ )
```

using assms **by** auto

lemma MMI_anbi12i: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**
A2: ch $\longleftrightarrow \vartheta$
shows ($\varphi \wedge$ ch) $\longleftrightarrow$ ($\psi \wedge \vartheta$)
using assms **by** auto

lemma MMI_keepel: **assumes** A1: A $\in$ C **and**
A2: B $\in$ C
shows if (φ , A , B) $\in$ C
using assms **by** auto

lemma MMI_imbi2d: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ (($\vartheta \longrightarrow \psi$) $\longleftrightarrow$ ($\vartheta \longrightarrow$ ch))
using assms **by** auto

lemma MMI_eqeltr: **assumes** A = B **and** B $\in$ C
shows A $\in$ C **using** assms **by** auto

lemma MMI_3impia: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ (ch $\longrightarrow \vartheta$)
shows ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_eqneqd: **assumes** A1: $\varphi \longrightarrow$ (A = B $\longleftrightarrow$ C = D)
shows $\varphi \longrightarrow$ (A $\neq$ B $\longleftrightarrow$ C $\neq$ D)
using assms **by** auto

lemma MMI_3ad2ant2: **assumes** A1: $\varphi \longrightarrow$ ch
shows ($\psi \wedge \varphi \wedge \vartheta$) $\longrightarrow$ ch
using assms **by** auto

lemma MMI_mp3anl3: **assumes** A1: ch **and**
A2: (($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta$) $\longrightarrow \tau$
shows (($\varphi \wedge \psi$) $\wedge \vartheta$) $\longrightarrow \tau$
using assms **by** auto

lemma MMI_bitr4d: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow \vartheta$)
using assms **by** auto

lemma `MMI_neeq1d`: **assumes** A1: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ (A $\neq$ C $\longleftrightarrow$ B $\neq$ C)
using assms **by** auto

lemma `MMI_3anim123i`: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch $\longrightarrow \vartheta$ **and**
A3: $\tau \longrightarrow \eta$
shows ($\varphi \wedge$ ch $\wedge \tau$) $\longrightarrow$ ($\psi \wedge \vartheta \wedge \eta$)
using assms **by** auto

lemma `MMI_3exp`: **assumes** A1: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow \vartheta$))
using assms **by** auto

lemma `MMI_exp4a`: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ((ch $\wedge \vartheta$) $\longrightarrow \tau$))
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow$ ($\vartheta \longrightarrow \tau$)))
using assms **by** auto

lemma `MMI_3imp1`: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow$ ($\vartheta \longrightarrow \tau$)))

shows (($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta$) $\longrightarrow \tau$
using assms **by** auto

lemma `MMI_anim1i`: **assumes** A1: $\varphi \longrightarrow \psi$
shows ($\varphi \wedge$ ch) $\longrightarrow$ ($\psi \wedge$ ch)
using assms **by** auto

lemma `MMI_3adantl1`: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
shows (($\tau \wedge \varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
using assms **by** auto

lemma `MMI_3adantl2`: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
shows (($\varphi \wedge \tau \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
using assms **by** auto

lemma `MMI_3comr`: **assumes** A1: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows (ch $\wedge \varphi \wedge \psi$) $\longrightarrow \vartheta$
using assms **by** auto

lemma `MMI_bitr3`: **assumes** A1: $\psi \longleftrightarrow \varphi$ **and**
A2: $\psi \longleftrightarrow$ ch
shows $\varphi \longleftrightarrow$ ch
using assms **by** auto

lemma MMI_anbi12d: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) **and**
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
shows $\varphi \longrightarrow ((\psi \wedge \vartheta) \longleftrightarrow$ (ch $\wedge \tau))$
using assms **by** auto

lemma MMI_pm3_26i: **assumes** A1: $\varphi \wedge \psi$
shows φ
using assms **by** auto

lemma MMI_pm3_27i: **assumes** A1: $\varphi \wedge \psi$
shows ψ
using assms **by** auto

lemma MMI_anabsan: **assumes** A1: $((\varphi \wedge \varphi) \wedge \psi) \longrightarrow$ ch
shows $(\varphi \wedge \psi) \longrightarrow$ ch
using assms **by** auto

lemma MMI_3eqtr4rd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ C = A **and**
A3: $\varphi \longrightarrow$ D = B
shows $\varphi \longrightarrow$ D = C
using assms **by** auto

lemma MMI_syl3an1: **assumes** A1: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \varphi$
shows ($\tau \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_syl3anl2: **assumes** A1: (($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta) \longrightarrow \tau$ **and**
A2: $\eta \longrightarrow \psi$
shows (($\varphi \wedge \eta \wedge$ ch) $\wedge \vartheta) \longrightarrow \tau$
using assms **by** auto

lemma MMI_jca: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow$ ch
shows $\varphi \longrightarrow (\psi \wedge$ ch)
using assms **by** auto

lemma MMI_3ad2ant3: **assumes** A1: $\varphi \longrightarrow$ ch
shows $(\psi \wedge \vartheta \wedge \varphi) \longrightarrow$ ch
using assms **by** auto

lemma MMI_anim2i: **assumes** A1: $\varphi \longrightarrow \psi$
shows (ch $\wedge \varphi) \longrightarrow$ (ch $\wedge \psi)$

```
  using assms by auto

lemma MMI_ancom:
  shows ( φ ∧ ψ ) ⟷ ( ψ ∧ φ )
 by auto

lemma MMI_anbi1i: assumes Aaa: φ ⟷ ψ
  shows ( φ ∧ ch ) ⟷ ( ψ ∧ ch )
  using assms by auto

lemma MMI_an42:
  shows ( ( φ ∧ ψ ) ∧ ( ch ∧ ϑ ) ) ⟷
 ( ( φ ∧ ch ) ∧ ( ϑ ∧ ψ ) )
 by auto

lemma MMI_sylanb: assumes A1: ( φ ∧ ψ ) ⟶ ch and
   A2: ϑ ⟷ φ
  shows ( ϑ ∧ ψ ) ⟶ ch
  using assms by auto

lemma MMI_an4:
  shows ( ( φ ∧ ψ ) ∧ ( ch ∧ ϑ ) ) ⟷
 ( ( φ ∧ ch ) ∧ ( ψ ∧ ϑ ) )
 by auto

lemma MMI_syl2anb: assumes A1: ( φ ∧ ψ ) ⟶ ch and
   A2: ϑ ⟷ φ and
   A3: τ ⟷ ψ
  shows ( ϑ ∧ τ ) ⟶ ch
  using assms by auto

lemma MMI_eqtr2d: assumes A1: φ ⟶ A = B and
   A2: φ ⟶ B = C
  shows φ ⟶ C = A
  using assms by auto

lemma MMI_sylbid: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
   A2: φ ⟶ ( ch ⟶ ϑ )
  shows φ ⟶ ( ψ ⟶ ϑ )
  using assms by auto

lemma MMI_sylanl1: assumes A1: ( ( φ ∧ ψ ) ∧ ch ) ⟶ ϑ and
   A2: τ ⟶ φ
  shows ( ( τ ∧ ψ ) ∧ ch ) ⟶ ϑ
  using assms by auto

lemma MMI_sylan2b: assumes A1: ( φ ∧ ψ ) ⟶ ch and
   A2: ϑ ⟷ ψ
  shows ( φ ∧ ϑ ) ⟶ ch
```

```
  using assms by auto

lemma MMI_pm3_22:
  shows ( φ ∧ ψ ) ⟶ ( ψ ∧ φ )
 by auto

lemma MMI_ancli: assumes A1: φ ⟶ ψ
  shows φ ⟶ ( φ ∧ ψ )
  using assms by auto

lemma MMI_ad2antlr: assumes A1: φ ⟶ ψ
  shows ( ( ch ∧ φ ) ∧ ϑ ) ⟶ ψ
  using assms by auto

lemma MMI_biimpa: assumes A1: φ ⟶ ( ψ ⟷ ch )
  shows ( φ ∧ ψ ) ⟶ ch
  using assms by auto

lemma MMI_sylan2i: assumes A1: φ ⟶ ( ( ψ ∧ ch ) ⟶ ϑ ) and
   A2: τ ⟶ ch
  shows φ ⟶ ( ( ψ ∧ τ ) ⟶ ϑ )
  using assms by auto

lemma MMI_3jca: assumes A1: φ ⟶ ψ and
   A2: φ ⟶ ch and
   A3: φ ⟶ ϑ
  shows φ ⟶ ( ψ ∧ ch ∧ ϑ )
  using assms by auto

lemma MMI_com34: assumes A1: φ ⟶ ( ψ ⟶ ( ch ⟶ ( ϑ ⟶ τ ) ) )

  shows φ ⟶ ( ψ ⟶ ( ϑ ⟶ ( ch ⟶ τ ) ) )
  using assms by auto

lemma MMI_imp43: assumes A1: φ ⟶ ( ψ ⟶ ( ch ⟶ ( ϑ ⟶ τ ) ) )

  shows ( ( φ ∧ ψ ) ∧ ( ch ∧ ϑ ) ) ⟶ τ
  using assms by auto

lemma MMI_3anass:
  shows ( φ ∧ ψ ∧ ch ) ⟷ ( φ ∧ ( ψ ∧ ch ) )
 by auto



lemma MMI_3eqtr4r: assumes A1: A = B and
   A2: C = A and
   A3: D = B
  shows D = C
```

```
  using assms by auto

lemma MMI_jctl: assumes A1: ψ
  shows φ ⟶ ( ψ ∧ φ )
  using assms by auto

lemma MMI_sylibr: assumes A1: φ ⟶ ψ and
   A2: ch ⟷ ψ
  shows φ ⟶ ch
  using assms by auto

lemma MMI_mpanl1: assumes A1: φ and
   A2: ( ( φ ∧ ψ ) ∧ ch ) ⟶ ϑ
  shows ( ψ ∧ ch ) ⟶ ϑ
  using assms by auto

lemma MMI_a1i: assumes A1: φ
  shows ψ ⟶ φ
  using assms by auto

lemma (in MMIsar0) MMI_opreqan12rd: assumes A1: φ ⟶ A = B and
   A2: ψ ⟶ C = D
  shows
 ( ψ ∧ φ ) ⟶ ( A + C ) = ( B + D )
 ( ψ ∧ φ ) ⟶ ( A · C ) = ( B · D )
 ( ψ ∧ φ ) ⟶ ( A - C ) = ( B - D )
 ( ψ ∧ φ ) ⟶ ( A / C ) = ( B / D )
  using assms by auto

lemma MMI_3adantl3: assumes A1: ( ( φ ∧ ψ ) ∧ ch ) ⟶ ϑ
  shows ( ( φ ∧ ψ ∧ τ ) ∧ ch ) ⟶ ϑ
  using assms by auto

lemma MMI_sylbi: assumes A1: φ ⟷ ψ and
   A2: ψ ⟶ ch
  shows φ ⟶ ch
  using assms by auto


lemma MMI_eirr:
  shows ¬ ( A ∈ A )
 by (rule mem_not_refl)

lemma MMI_eleq1i: assumes A1: A = B
  shows A ∈ C ⟷ B ∈ C
  using assms by auto

lemma MMI_mtbir: assumes A1: ¬ ( ψ ) and
```

```
    A2: φ ⟷ ψ
   shows ¬ ( φ )
   using assms by auto

lemma MMI_mto: assumes A1: ¬ ( ψ ) and
    A2: φ ⟶ ψ
   shows ¬ ( φ )
   using assms by auto

lemma MMI_df_nel:
   shows ( A ∉ B ⟷ ¬ ( A ∈ B ) )
  by auto

lemma MMI_snid: assumes A1: A isASet
   shows A ∈ { A }
   using assms by auto

lemma MMI_en2lp:
   shows ¬ ( A ∈ B ∧ B ∈ A )
proof
  assume A1: A ∈ B ∧ B ∈ A
  then have A ∈ B by simp
  moreover
  { assume ¬ (¬ ( A ∈ B ∧ B ∈ A ))
    then have B∈A by auto}
  ultimately have ¬( A ∈ B ∧ B ∈ A )
    by (rule mem_asym)
  with A1 show False by simp
qed

lemma MMI_imnan:
   shows ( φ ⟶ ¬ ( ψ ) ) ⟷ ¬ ( ( φ ∧ ψ ) )
  by auto



lemma MMI_sseqtr4: assumes A1: A ⊆ B and
    A2: C = B
   shows A ⊆ C
   using assms by auto

lemma MMI_ssun1:
   shows A ⊆ ( A ∪ B )
  by auto

lemma MMI_ibar:
   shows φ ⟶ ( ψ ⟷ ( φ ∧ ψ ) )
  by auto
```

```
lemma MMI_mtbiri: assumes Amin: ¬ ( ch ) and
    Amaj: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ¬ ( ψ )
   using assms by auto

lemma MMI_con2i: assumes Aa: φ ⟶ ¬ ( ψ )
   shows ψ ⟶ ¬ ( φ )
   using assms by auto

lemma MMI_intnand: assumes A1: φ ⟶ ¬ ( ψ )
   shows φ ⟶ ¬ ( ( ch ∧ ψ ) )
   using assms by auto

lemma MMI_intnanrd: assumes A1: φ ⟶ ¬ ( ψ )
   shows φ ⟶ ¬ ( ( ψ ∧ ch ) )
   using assms by auto

lemma MMI_biorf:
   shows ¬ ( φ ) ⟶ ( ψ ⟷ ( φ ∨ ψ ) )
  by auto

lemma MMI_bitr2d: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: φ ⟶ ( ch ⟷ ϑ )
   shows φ ⟶ ( ϑ ⟷ ψ )
   using assms by auto

lemma MMI_orass:
   shows ( ( φ ∨ ψ ) ∨ ch ) ⟷ ( φ ∨ ( ψ ∨ ch ) )
  by auto

lemma MMI_orcom:
   shows ( φ ∨ ψ ) ⟷ ( ψ ∨ φ )
  by auto



lemma MMI_3bitr4d: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: φ ⟶ ( ϑ ⟷ ψ ) and
    A3: φ ⟶ ( τ ⟷ ch )
   shows φ ⟶ ( ϑ ⟷ τ )
   using assms by auto

lemma MMI_3imtr4d: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: φ ⟶ ( ϑ ⟷ ψ ) and
    A3: φ ⟶ ( τ ⟷ ch )
   shows φ ⟶ ( ϑ ⟶ τ )
   using assms by auto
```

```
lemma MMI_3impdi: assumes A1: ( ( φ ∧ ψ ) ∧ ( φ ∧ ch ) ) ⟶ ϑ
   shows ( φ ∧ ψ ∧ ch ) ⟶ ϑ
   using assms by auto

lemma MMI_bi2anan9: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: ϑ ⟶ ( τ ⟷ η )
   shows ( φ ∧ ϑ ) ⟶ ( ( ψ ∧ τ ) ⟷ ( ch ∧ η ) )
   using assms by auto

lemma MMI_ssel2:
   shows ( ( A ⊆ B ∧ C ∈ A ) ⟶ C ∈ B )
  by auto

lemma MMI_an1rs: assumes A1: ( ( φ ∧ ψ ) ∧ ch ) ⟶ ϑ
   shows ( ( φ ∧ ch ) ∧ ψ ) ⟶ ϑ
   using assms by auto



lemma MMI_ralbidva: assumes A1: ∀x. ( φ ∧ x ∈ A ) ⟶ ( ψ(x) ⟷ ch(x)
)
   shows φ ⟶  ( ( ∀ x ∈ A . ψ(x) ) ⟷ ( ∀ x ∈ A . ch(x) ) )
   using assms by auto

lemma MMI_rexbidva: assumes A1: ∀x. ( φ ∧ x ∈ A ) ⟶ ( ψ(x) ⟷ ch(x)
)
   shows φ ⟶  ( ( ∃ x ∈ A . ψ(x) ) ⟷ ( ∃ x ∈ A . ch(x) ) )
   using assms by auto

lemma MMI_con2bid: assumes A1: φ ⟶ ( ψ ⟷ ¬ ( ch ) )
   shows φ ⟶ ( ch ⟷ ¬ ( ψ ) )
   using assms by auto


lemma MMI_so: assumes
  A1: ∀x y z. ( x ∈ A ∧ y ∈ A ∧ z ∈ A ) ⟶
  ( ( ⟨x,y⟩ ∈ R ⟷ ¬ ( ( x = y ∨ ⟨y, x⟩ ∈ R ) ) ) ∧
  ( ( ⟨x, y⟩ ∈ R  ∧ ⟨y, z⟩ ∈ R ) ⟶ ⟨x, z⟩ ∈ R ) )
  shows R Orders A
  using assms StrictOrder_def by auto



lemma MMI_con1bid: assumes A1: φ ⟶ ( ¬ ( ψ ) ⟷ ch )
   shows φ ⟶ ( ¬ ( ch ) ⟷ ψ )
   using assms by auto
```

```
lemma MMI_sotrieq:
  shows ( (R Orders A) ∧ ( B ∈ A ∧ C ∈ A ) ) ⟶
  ( B = C ⟷ ¬ ( ( ⟨B,C⟩ ∈ R ∨ ⟨C, B⟩ ∈ R ) ) )
proof -
  { assume A1: R Orders A  and A2: B ∈ A ∧ C ∈ A
    from A1 have ∀x y z. (x∈A ∧ y∈A ∧ z∈A) ⟶
      ((⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R)) ∧
      ((⟨x,y⟩ ∈ R ∧ ⟨y,z⟩ ∈ R ⟶ ⟨x,z⟩ ∈ R)
      by (unfold StrictOrder_def)
    then have
      ∀x y. x∈A ∧ y∈A ⟶ ((⟨x,y⟩ ∈ R ⟷ ¬(x=y ∨ ⟨y,x⟩ ∈ R))
      by auto
    with A2 have I: ⟨B,C⟩ ∈ R ⟷ ¬(B=C ∨ ⟨C,B⟩ ∈ R)
      by blast
    then have B = C ⟷ ¬ ( ⟨B,C⟩ ∈ R ∨ ⟨C, B⟩ ∈ R )
      by auto
  } then show ( (R Orders A) ∧ ( B ∈ A ∧ C ∈ A ) ) ⟶
      ( B = C ⟷ ¬ ( ( ⟨B,C⟩ ∈ R ∨ ⟨C, B⟩ ∈ R ) ) ) by simp
qed

lemma MMI_bicomd: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ch ⟷ ψ )
   using assms by auto

lemma MMI_sotrieq2:
  shows ( R Orders A ∧ ( B ∈ A ∧ C ∈ A ) ) ⟶
  ( B =  C ⟷ ( ¬ ( ⟨B, C⟩ ∈ R ) ∧ ¬ ( ⟨C, B⟩ ∈ R ) ) )
  using  MMI_sotrieq by auto

lemma MMI_orc:
   shows φ ⟶ ( φ ∨ ψ )
  by auto

lemma MMI_syl6bbr: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: ϑ ⟷ ch
   shows φ ⟶ ( ψ ⟷ ϑ )
   using assms by auto



lemma MMI_orbi1i: assumes A1: φ ⟷ ψ
   shows ( φ ∨ ch ) ⟷ ( ψ ∨ ch )
   using assms by auto

lemma MMI_syl5rbbr: assumes A1: φ ⟶ ( ψ ⟷ ch ) and
    A2: ψ ⟷ ϑ
   shows φ ⟶ ( ch ⟷ ϑ )
   using assms by auto
```

```
lemma MMI_anbi2d: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ( ϑ ∧ ψ ) ⟷ ( ϑ ∧ ch ) )
   using assms by auto

lemma MMI_ord: assumes A1: φ ⟶ ( ψ ∨ ch )
   shows φ ⟶ ( ¬ ( ψ ) ⟶ ch )
   using assms by auto

lemma MMI_impbid: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: φ ⟶ ( ch ⟶ ψ )
   shows φ ⟶ ( ψ ⟷ ch )
   using assms by blast

lemma MMI_jcad: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: φ ⟶ ( ψ ⟶ ϑ )
   shows φ ⟶ ( ψ ⟶ ( ch ∧ ϑ ) )
   using assms by auto

lemma MMI_ax_1:
   shows φ ⟶ ( ψ ⟶ φ )
  by auto

lemma MMI_pm2_24:
   shows φ ⟶ ( ¬ ( φ ) ⟶ ψ )
  by auto

lemma MMI_imp3a: assumes A1: φ ⟶ ( ψ ⟶ ( ch ⟶ ϑ ) )
   shows φ ⟶ ( ( ψ ∧ ch ) ⟶ ϑ )
   using assms by auto

lemma (in MMIsar0) MMI_breq1:
   shows
  A = B ⟶ ( A ≤ C ⟷ B ≤ C )
  A = B ⟶ ( A < C ⟷ B < C )
  by auto

lemma MMI_biimprd: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ( ch ⟶ ψ )
   using assms by auto

lemma MMI_jaod: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: φ ⟶ ( ϑ ⟶ ch )
   shows φ ⟶ ( ( ψ ∨ ϑ ) ⟶ ch )
   using assms by auto

lemma MMI_com23: assumes A1: φ ⟶ ( ψ ⟶ ( ch ⟶ ϑ ) )
   shows φ ⟶ ( ch ⟶ ( ψ ⟶ ϑ ) )
   using assms by auto
```

```
lemma (in MMIsar0) MMI_breq2:
   shows
  A = B ⟶ ( C ≤ A ⟷ C ≤ B )
  A = B ⟶ ( C < A ⟷ C < B )
  by auto

lemma MMI_syld: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: φ ⟶ ( ch ⟶ ϑ )
   shows φ ⟶ ( ψ ⟶ ϑ )
   using assms by auto

lemma MMI_biimpcd: assumes A1: φ ⟶ ( ψ ⟷ ch )
   shows ψ ⟶ ( φ ⟶ ch )
   using assms by auto

lemma MMI_mp2and: assumes A1: φ ⟶ ψ and
    A2: φ ⟶ ch and
    A3: φ ⟶ ( ( ψ ∧ ch ) ⟶ ϑ )
   shows φ ⟶ ϑ
   using assms by auto



lemma MMI_sonr:
   shows ( R Orders A ∧ B ∈ A ) ⟶ ¬ ( ⟨B,B⟩ ∈ R )
  unfolding StrictOrder_def by auto

lemma MMI_orri: assumes A1: ¬ ( φ ) ⟶ ψ
   shows φ ∨ ψ
   using assms by auto

lemma MMI_mpbiri: assumes Amin: ch and
    Amaj: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ψ
   using assms by auto

lemma MMI_pm2_46:
   shows ¬ ( ( φ ∨ ψ ) ) ⟶ ¬ ( ψ )
  by auto

lemma MMI_elun:
   shows A ∈ ( B ∪ C ) ⟷ ( A ∈ B ∨ A ∈ C )
  by auto

lemma (in MMIsar0) MMI_pnfxr:
   shows +∞ ∈ ℝ*
  using cxr_def by simp

lemma MMI_elisseti: assumes A1: A ∈ B
```

```
   shows A isASet
   using assms by auto

lemma (in MMIsar0) MMI_mnfxr:
   shows −∞ ∈ ℝ*
  using  cxr_def by simp

lemma MMI_elpr2: assumes A1: B isASet and
    A2: C isASet
   shows A ∈ { B , C } ⟷ ( A = B ∨ A = C )
   using assms by auto

lemma MMI_orbi2i: assumes A1: φ ⟷ ψ
   shows ( ch ∨ φ ) ⟷ ( ch ∨ ψ )
   using assms by auto

lemma MMI_3orass:
   shows ( φ ∨ ψ ∨ ch ) ⟷ ( φ ∨ ( ψ ∨ ch ) )
  by auto

lemma MMI_bitr4: assumes A1: φ ⟷ ψ and
    A2: ch ⟷ ψ
   shows φ ⟷ ch
   using assms by auto

lemma MMI_eleq2:
   shows A = B ⟶ ( C ∈ A ⟷ C ∈ B )
  by auto

lemma MMI_nelneq:
   shows ( A ∈ C ∧ ¬ ( B ∈ C ) ) ⟶ ¬ ( A = B )
  by auto



lemma MMI_df_pr:
   shows { A , B } = ( { A } ∪ { B } )
  by auto

lemma MMI_ineq2i: assumes A1: A = B
   shows ( C ∩ A ) = ( C ∩ B )
   using assms by auto

lemma MMI_mt2: assumes A1: ψ and
    A2: φ ⟶ ¬ ( ψ )
   shows ¬ ( φ )
   using assms by auto

lemma MMI_disjsn:
```

```
   shows ( A ∩ { B } ) = 0 ⟷ ¬ ( B ∈ A )
  by auto

lemma MMI_undisj2:
   shows ( ( A ∩ B ) =
 0 ∧ ( A ∩ C ) =
 0 ) ⟷ ( A ∩ ( B ∪ C ) ) = 0
  by auto

lemma MMI_disjssun:
   shows ( ( A ∩ B ) =  0 ⟶ ( A ⊆ ( B ∪ C ) ⟷ A ⊆ C ) )
  by auto

lemma MMI_uncom:
   shows ( A ∪ B ) = ( B ∪ A )
  by auto

lemma MMI_sseq2i: assumes A1: A = B
   shows ( C ⊆ A ⟷ C ⊆ B )
   using assms by auto

lemma MMI_disj:
   shows ( A ∩ B ) =
 0 ⟷ ( ∀ x ∈ A . ¬ ( x ∈ B ) )
  by auto

lemma MMI_syl5ibr: assumes A1: φ ⟶ ( ψ ⟶ ch ) and
    A2: ψ ⟷ ϑ
   shows φ ⟶ ( ϑ ⟶ ch )
   using assms by auto

lemma MMI_con3d: assumes A1: φ ⟶ ( ψ ⟶ ch )
   shows φ ⟶ ( ¬ ( ch ) ⟶ ¬ ( ψ ) )
   using assms by auto



lemma MMI_dfrex2:
  shows ( ∃ x ∈ A . φ(x) ) ⟷  ¬ ( ( ∀ x ∈ A . ¬ φ(x) ) )
  by auto

lemma MMI_visset:
   shows x isASet
  by auto

lemma MMI_elpr: assumes A1: A isASet
   shows A ∈ { B , C } ⟷ ( A = B ∨ A = C )
   using assms by auto
```

lemma MMI_rexbii: **assumes** A1: $\forall$x. φ(x) $\longleftrightarrow$ ψ(x)
  shows ($\exists$ x $\in$ A . φ(x)) $\longleftrightarrow$ ($\exists$ x $\in$ A . ψ(x))
  using assms **by** auto

lemma MMI_r19_43:
  shows ($\exists$ x $\in$ A . (φ(x) $\vee$ ψ(x))) $\longleftrightarrow$
 (($\exists$ x $\in$ A . φ(x) $\vee$ ($\exists$ x $\in$ A . ψ(x))))
 by auto

lemma MMI_exancom:
  shows ($\exists$ x . (φ(x) $\wedge$ ψ(x))) $\longleftrightarrow$
 ($\exists$ x . (ψ(x) $\wedge$ φ(x)))
 by auto

lemma MMI_ceqsexv: **assumes** A1: A isASet **and**
  A2: $\forall$x. x = A $\longrightarrow$ (φ(x) $\longleftrightarrow$ ψ(x))
  shows ($\exists$ x . (x = A $\wedge$ φ(x))) $\longleftrightarrow$ ψ(A)
  using assms **by** auto

lemma MMI_orbi12i_orig: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
  A2: ch $\longleftrightarrow$ ϑ
  shows (φ $\vee$ ch) $\longleftrightarrow$ (ψ $\vee$ ϑ)
  using assms **by** auto

lemma MMI_orbi12i: **assumes** A1: ($\exists$x. φ(x)) $\longleftrightarrow$ ψ **and**
  A2: ($\exists$x. ch(x)) $\longleftrightarrow$ ϑ
  shows ($\exists$x. φ(x)) $\vee$ ($\exists$x. ch(x)) $\longleftrightarrow$ (ψ $\vee$ ϑ)
  using assms **by** auto

lemma MMI_syl6ib: **assumes** A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch) **and**
  A2: ch $\longleftrightarrow$ ϑ
  shows φ $\longrightarrow$ (ψ $\longrightarrow$ ϑ)
  using assms **by** auto

lemma MMI_intnan: **assumes** A1: $\neg$ (φ)
  shows $\neg$ ((ψ $\wedge$ φ))
  using assms **by** auto

lemma MMI_intnanr: **assumes** A1: $\neg$ (φ)
  shows $\neg$ ((φ $\wedge$ ψ))
  using assms **by** auto

lemma MMI_pm3_2ni: **assumes** A1: $\neg$ (φ) **and**
  A2: $\neg$ (ψ)
  shows $\neg$ ((φ $\vee$ ψ))
  using assms **by** auto

lemma (**in** MMIsar0) MMI_breq12:
  shows

```
  ( A = B ∧ C = D ) ⟶ ( A < C ⟷ B < D )
  ( A = B ∧ C = D ) ⟶ ( A ≤ C ⟷ B ≤ D )
  by auto

lemma MMI_necom:
   shows A ≠ B ⟷ B ≠ A
  by auto

lemma MMI_3jaoi: assumes A1: φ ⟶ ψ and
    A2: ch ⟶ ψ and
    A3: ϑ ⟶ ψ
   shows ( φ ∨ ch ∨ ϑ ) ⟶ ψ
   using assms by auto

lemma MMI_jctr: assumes A1: ψ
   shows φ ⟶ ( φ ∧ ψ )
   using assms by auto

lemma MMI_olc:
   shows φ ⟶ ( ψ ∨ φ )
  by auto

lemma MMI_3syl: assumes A1: φ ⟶ ψ and
    A2: ψ ⟶ ch and
    A3: ch ⟶ ϑ
   shows φ ⟶ ϑ
   using assms by auto



lemma MMI_mtbird: assumes Amin: φ ⟶ ¬ ( ch ) and
    Amaj: φ ⟶ ( ψ ⟷ ch )
   shows φ ⟶ ¬ ( ψ )
   using assms by auto

lemma MMI_pm2_21d: assumes A1: φ ⟶ ¬ ( ψ )
   shows φ ⟶ ( ψ ⟶ ch )
   using assms by auto

lemma MMI_3jaodan: assumes A1: ( φ ∧ ψ ) ⟶ ch and
    A2: ( φ ∧ ϑ ) ⟶ ch and
    A3: ( φ ∧ τ ) ⟶ ch
   shows ( φ ∧ ( ψ ∨ ϑ ∨ τ ) ) ⟶ ch
   using assms by auto

lemma MMI_sylan2br: assumes A1: ( φ ∧ ψ ) ⟶ ch and
    A2: ψ ⟷ ϑ
   shows ( φ ∧ ϑ ) ⟶ ch
   using assms by auto
```

lemma MMI_3jaoian: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch **and**
A2: (ϑ $\wedge$ ψ) $\longrightarrow$ ch **and**
A3: (τ $\wedge$ ψ) $\longrightarrow$ ch
shows ((φ $\vee$ ϑ $\vee$ τ) $\wedge$ ψ) $\longrightarrow$ ch
using assms **by** auto

lemma MMI_mtbid: **assumes** Amin: φ $\longrightarrow$ $\neg$ (ψ) **and**
Amaj: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ $\neg$ (ch)
using assms **by** auto

lemma MMI_con1d: **assumes** A1: φ $\longrightarrow$ ($\neg$ (ψ) $\longrightarrow$ ch)
shows φ $\longrightarrow$ ($\neg$ (ch) $\longrightarrow$ ψ)
using assms **by** auto

lemma MMI_pm2_21nd: **assumes** A1: φ $\longrightarrow$ ψ
shows φ $\longrightarrow$ ($\neg$ (ψ) $\longrightarrow$ ch)
using assms **by** auto

lemma MMI_syl3an1b: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ **and**
A2: τ $\longleftrightarrow$ φ
shows (τ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
using assms **by** auto

lemma MMI_adantld: **assumes** A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch)
shows φ $\longrightarrow$ ((ϑ $\wedge$ ψ) $\longrightarrow$ ch)
using assms **by** auto

lemma MMI_adantrd: **assumes** A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch)
shows φ $\longrightarrow$ ((ψ $\wedge$ ϑ) $\longrightarrow$ ch)
using assms **by** auto

lemma MMI_anasss: **assumes** A1: ((φ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ ϑ
shows (φ $\wedge$ (ψ $\wedge$ ch)) $\longrightarrow$ ϑ
using assms **by** auto

lemma MMI_syl3an3b: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ **and**
A2: τ $\longleftrightarrow$ ch
shows (φ $\wedge$ ψ $\wedge$ τ) $\longrightarrow$ ϑ
using assms **by** auto

lemma MMI_mpbid: **assumes** Amin: φ $\longrightarrow$ ψ **and**
Amaj: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ch
using assms **by** auto

lemma MMI_orbi12d: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) **and**
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
shows $\varphi \longrightarrow ((\psi \vee \vartheta) \longleftrightarrow ($ ch $\vee \tau))$
using assms **by** auto

lemma MMI_ianor:
shows $\neg (\varphi \wedge \psi) \longleftrightarrow \neg \varphi \vee \neg \psi$
by auto

lemma MMI_bitr2: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**
A2: $\psi \longleftrightarrow$ ch
shows ch $\longleftrightarrow \varphi$
using assms **by** auto

lemma MMI_biimp: **assumes** A1: $\varphi \longleftrightarrow \psi$
shows $\varphi \longrightarrow \psi$
using assms **by** auto

lemma MMI_mpan2d: **assumes** A1: $\varphi \longrightarrow$ ch **and**
A2: $\varphi \longrightarrow ((\psi \wedge$ ch $) \longrightarrow \vartheta)$
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
using assms **by** auto

lemma MMI_ad2antrr: **assumes** A1: $\varphi \longrightarrow \psi$
shows $((\varphi \wedge$ ch $) \wedge \vartheta) \longrightarrow \psi$
using assms **by** auto

lemma MMI_biimpac: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $(\psi \wedge \varphi) \longrightarrow$ ch
using assms **by** auto

lemma MMI_con2bii: **assumes** A1: $\varphi \longleftrightarrow \neg (\psi)$
shows $\psi \longleftrightarrow \neg (\varphi)$
using assms **by** auto

lemma MMI_pm3_26bd: **assumes** A1: $\varphi \longleftrightarrow (\psi \wedge$ ch)
shows $\varphi \longrightarrow \psi$
using assms **by** auto

lemma MMI_biimpr: **assumes** A1: $\varphi \longleftrightarrow \psi$
shows $\psi \longrightarrow \varphi$
using assms **by** auto

lemma (**in** MMIsar0) MMI_3brtr3g: **assumes** A1: $\varphi \longrightarrow$ A < B **and**

A2: A = C **and**
A3: B = D
shows $\varphi \longrightarrow$ C < D
using assms **by** auto

lemma (**in** MMIsar0) MMI_breq12i: **assumes** A1: A = B **and**
A2: C = D
shows
A < C $\longleftrightarrow$ B < D
A $\leq$ C $\longleftrightarrow$ B $\leq$ D
using assms **by** auto

lemma MMI_negbii: **assumes** Aa: $\varphi \longleftrightarrow \psi$
shows $\neg\varphi \longleftrightarrow \neg\psi$
using assms **by** auto

lemma (**in** MMIsar0) MMI_breq1i: **assumes** A1: A = B
shows
A < C $\longleftrightarrow$ B < C
A $\leq$ C $\longleftrightarrow$ B $\leq$ C
using assms **by** auto

lemma MMI_syl5eqr: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: A = C
shows $\varphi \longrightarrow$ C = B
using assms **by** auto

lemma (**in** MMIsar0) MMI_breq2d: **assumes** A1: $\varphi \longrightarrow$ A = B
shows
$\varphi \longrightarrow$ C < A $\longleftrightarrow$ C < B
$\varphi \longrightarrow$ C $\leq$ A $\longleftrightarrow$ C $\leq$ B
using assms **by** auto

lemma MMI_ccase: **assumes** A1: $\varphi \wedge \psi \longrightarrow \tau$ **and**
A2: ch $\wedge\ \psi \longrightarrow \tau$ **and**
A3: $\varphi \wedge \vartheta \longrightarrow \tau$ **and**
A4: ch $\wedge\ \vartheta \longrightarrow \tau$
shows ($\varphi \vee$ ch) $\wedge$ ($\psi \vee \vartheta$) $\longrightarrow \tau$
using assms **by** auto

lemma MMI_pm3_27bd: **assumes** A1: $\varphi \longleftrightarrow \psi \wedge$ ch
shows $\varphi \longrightarrow$ ch
using assms **by** auto

lemma MMI_nsyl3: **assumes** A1: $\varphi \longrightarrow \neg\psi$ **and**

A2: ch $\longrightarrow$ ψ
shows ch $\longrightarrow$ $\neg\varphi$
using assms **by** auto

lemma MMI_jctild: **assumes** A1: φ $\longrightarrow$ ψ $\longrightarrow$ ch **and**
A2: φ $\longrightarrow$ ϑ
shows φ $\longrightarrow$
ψ $\longrightarrow$ ϑ $\wedge$ ch
using assms **by** auto

lemma MMI_jctird: **assumes** A1: φ $\longrightarrow$ ψ $\longrightarrow$ ch **and**
A2: φ $\longrightarrow$ ϑ
shows φ $\longrightarrow$
ψ $\longrightarrow$ ch $\wedge$ ϑ
using assms **by** auto

lemma MMI_ccase2: **assumes** A1: φ $\wedge$ ψ $\longrightarrow$ τ **and**
A2: ch $\longrightarrow$ τ **and**
A3: ϑ $\longrightarrow$ τ
shows (φ $\vee$ ch) $\wedge$ (ψ $\vee$ ϑ) $\longrightarrow$ τ
using assms **by** auto

lemma MMI_3bitr3r: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
A2: φ $\longleftrightarrow$ ch **and**
A3: ψ $\longleftrightarrow$ ϑ
shows ϑ $\longleftrightarrow$ ch
using assms **by** auto

lemma (**in** MMIsar0) MMI_syl6breq: **assumes** A1: φ $\longrightarrow$ A < B **and**
A2: B = C
shows
φ $\longrightarrow$ A < C
using assms **by** auto

lemma MMI_pm2_61i: **assumes** A1: φ $\longrightarrow$ ψ **and**
A2: $\neg\varphi$ $\longrightarrow$ ψ
shows ψ
using assms **by** auto

lemma MMI_syl6req: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: B = C
shows φ $\longrightarrow$ C = A
using assms **by** auto

lemma MMI_pm2_61d: **assumes** A1: φ $\longrightarrow$ ψ $\longrightarrow$ ch **and**

A2: $\varphi \longrightarrow$
$\neg\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms **by** auto

lemma MMI_orim1d: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \vee \vartheta \longrightarrow$ ch $\vee\ \vartheta$
using assms **by** auto

lemma (**in** MMIsar0) MMI_breq1d: **assumes** A1: $\varphi \longrightarrow$ A = B
shows
$\varphi \longrightarrow$ A < C $\longleftrightarrow$ B < C
$\varphi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ C
using assms **by** auto

lemma (**in** MMIsar0) MMI_breq12d: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ C = D
shows
$\varphi \longrightarrow$ A < C $\longleftrightarrow$ B < D
$\varphi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ D
using assms **by** auto

lemma MMI_bibi2d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
($\vartheta \longleftrightarrow \psi$) $\longleftrightarrow$
$\vartheta \longleftrightarrow$ ch
using assms **by** auto

lemma MMI_con4bid: **assumes** A1: $\varphi \longrightarrow$
$\neg\psi \longleftrightarrow \neg$ch
shows $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
using assms **by** auto

lemma MMI_3com13: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows ch $\wedge\ \psi \wedge \varphi \longrightarrow \vartheta$
using assms **by** auto

lemma MMI_3bitr3rd: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\varphi \longrightarrow$
$\psi \longleftrightarrow \vartheta$ **and**
A3: $\varphi \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$

$\tau \longleftrightarrow \vartheta$
using assms **by** auto

lemma MMI_3imtr4g: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**
A2: $\vartheta \longleftrightarrow \psi$ **and**
A3: $\tau \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\vartheta \longrightarrow \tau$
using assms **by** auto

lemma MMI_expcom: **assumes** A1: $\varphi \wedge \psi \longrightarrow$ ch
shows $\psi \longrightarrow \varphi \longrightarrow$ ch
using assms **by** auto

lemma (**in** MMIsar0) MMI_breq2i: **assumes** A1: A = B
shows
C < A $\longleftrightarrow$ C < B
C $\leq$ A $\longleftrightarrow$ C $\leq$ B
using assms **by** auto

lemma MMI_3bitr2r: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**
A2: ch $\longleftrightarrow \psi$ **and**
A3: ch $\longleftrightarrow \vartheta$
shows $\vartheta \longleftrightarrow \varphi$
using assms **by** auto

lemma MMI_dedth4h: **assumes** A1: A = if(φ, A, R) $\longrightarrow$
$\tau \longleftrightarrow \eta$ **and**
A2: B = if(ψ, B, S) $\longrightarrow$
$\eta \longleftrightarrow \zeta$ **and**
A3: C = if(ch, C, F) $\longrightarrow$
$\zeta \longleftrightarrow$ si **and**
A4: D = if(ϑ, D, G) $\longrightarrow$ si $\longleftrightarrow$ rh **and**
A5: rh
shows ($\varphi \wedge \psi$) $\wedge$ ch $\wedge$ $\vartheta \longrightarrow \tau$
using assms **by** auto

lemma MMI_anbi1d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \wedge \vartheta \longleftrightarrow$ ch $\wedge$ ϑ
using assms **by** auto

lemma (**in** MMIsar0) MMI_breqtrrd: **assumes** A1: $\varphi \longrightarrow$ A < B **and**
A2: $\varphi \longrightarrow$ C = B

shows $\varphi \longrightarrow$ A < C
using assms **by** auto

lemma MMI_syl3an: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \varphi$ **and**
A3: $\eta \longrightarrow \psi$ **and**
A4: $\zeta \longrightarrow$ ch
shows $\tau \wedge \eta \wedge \zeta \longrightarrow \vartheta$
using assms **by** auto

lemma MMI_3bitrd: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\varphi \longrightarrow$
ch $\longleftrightarrow \vartheta$ **and**
A3: $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\psi \longleftrightarrow \tau$
using assms **by** auto

lemma (**in** MMIsar0) MMI_breqtr: **assumes** A1: A < B **and**
A2: B = C
shows A < C
using assms **by** auto

lemma MMI_mpi: **assumes** A1: ψ **and**
A2: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms **by** auto

lemma MMI_eqtr2: **assumes** A1: A = B **and**
A2: B = C
shows C = A
using assms **by** auto

lemma MMI_eqneqi: **assumes** A1: A = B $\longleftrightarrow$ C = D
shows A $\neq$ B $\longleftrightarrow$ C $\neq$ D
using assms **by** auto

lemma (**in** MMIsar0) MMI_eqbrtrrd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ A < C
shows $\varphi \longrightarrow$ B < C
using assms **by** auto

lemma MMI_mpd: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms **by** auto

lemma MMI_mpdan: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \wedge \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
using assms **by** auto

lemma (**in** MMIsar0) MMI_breqtrd: **assumes** A1: $\varphi \longrightarrow$ A < B **and**
A2: $\varphi \longrightarrow$ B = C
shows $\varphi \longrightarrow$ A < C
using assms **by** auto

lemma MMI_mpand: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ch $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_imbi1d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$(\psi \longrightarrow \vartheta) \longleftrightarrow$
(ch $\longrightarrow \vartheta$)
using assms **by** auto

lemma MMI_mtbii: **assumes** Amin: $\neg\psi$ **and**
Amaj: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow \neg$ch
using assms **by** auto

lemma MMI_sylan2d: **assumes** A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$ **and**
A2: $\varphi \longrightarrow \tau \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \wedge \tau \longrightarrow \vartheta$
using assms **by** auto

lemma MMI_imp32: **assumes** A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$

shows $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
using assms by auto

lemma (in MMIsar0) MMI_breqan12d: assumes A1: $\varphi \longrightarrow$ A = B and
A2: $\psi \longrightarrow$ C = D
shows
$\varphi \wedge \psi \longrightarrow$ A < C $\longleftrightarrow$ B < D
$\varphi \wedge \psi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ D
using assms by auto

lemma MMI_a1dd: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \longrightarrow \vartheta \longrightarrow$ ch
using assms by auto

lemma (in MMIsar0) MMI_3brtr3d: assumes A1: $\varphi \longrightarrow$ A $\leq$ B and
A2: $\varphi \longrightarrow$ A = C and
A3: $\varphi \longrightarrow$ B = D
shows $\varphi \longrightarrow$ C $\leq$ D
using assms by auto

lemma MMI_ad2antll: assumes A1: $\varphi \longrightarrow \psi$
shows ch $\wedge \vartheta \wedge \varphi \longrightarrow \psi$
using assms by auto

lemma MMI_adantrrl: assumes A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge \tau \wedge$ ch $\longrightarrow \vartheta$
using assms by auto

lemma MMI_syl2ani: assumes A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$ and
A2: $\tau \longrightarrow \psi$ and
A3: $\eta \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\tau \wedge \eta \longrightarrow \vartheta$
using assms by auto

lemma MMI_im2anan9: assumes A1: $\varphi \longrightarrow \psi \longrightarrow$ ch and
A2: $\vartheta \longrightarrow$
$\tau \longrightarrow \eta$
shows $\varphi \wedge \vartheta \longrightarrow$
$\psi \wedge \tau \longrightarrow$ ch $\wedge \eta$
using assms by auto

lemma MMI_ancomsd: assumes A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$
ch $\wedge \psi \longrightarrow \vartheta$
using assms by auto

lemma MMI_mpani: **assumes** A1: ψ **and**
 A2: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ch $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_syldan: **assumes** A1: $\varphi \wedge \psi \longrightarrow$ ch **and**
 A2: $\varphi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \longrightarrow \vartheta$
using assms **by** auto

lemma MMI_mp3anl1: **assumes** A1: φ **and**
 A2: ($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta \longrightarrow \tau$
shows ($\psi \wedge$ ch) $\wedge \vartheta \longrightarrow \tau$
using assms **by** auto

lemma MMI_3ad2ant1: **assumes** A1: $\varphi \longrightarrow$ ch
shows $\varphi \wedge \psi \wedge \vartheta \longrightarrow$ ch
using assms **by** auto

lemma MMI_pm3_2:
shows $\varphi \longrightarrow$
$\psi \longrightarrow \varphi \wedge \psi$
by auto

lemma MMI_pm2_43i: **assumes** A1: $\varphi \longrightarrow$
$\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow \psi$
using assms **by** auto

lemma MMI_jctil: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
 A2: ch
shows $\varphi \longrightarrow$ ch $\wedge \psi$
using assms **by** auto

lemma MMI_mpanl12: **assumes** A1: φ **and**
 A2: ψ **and**
 A3: ($\varphi \wedge \psi$) $\wedge$ ch $\longrightarrow \vartheta$
shows ch $\longrightarrow \vartheta$
using assms **by** auto

lemma MMI_mpanr1: **assumes** A1: ψ **and**
 A2: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge$ ch $\longrightarrow \vartheta$

```
  using assms by auto

lemma MMI_ad2antrl: assumes A1: φ ⟶ ψ
  shows ch ∧ φ ∧ ϑ ⟶ ψ
  using assms by auto

lemma MMI_3adant3r: assumes A1: φ ∧ ψ ∧ ch ⟶ ϑ
  shows φ ∧ ψ ∧ ch ∧ τ ⟶ ϑ
  using assms by auto

lemma MMI_3adant1l: assumes A1: φ ∧ ψ ∧ ch ⟶ ϑ
  shows (τ ∧ φ) ∧ ψ ∧ ch ⟶ ϑ
  using assms by auto

lemma MMI_3adant2r: assumes A1: φ ∧ ψ ∧ ch ⟶ ϑ
  shows φ ∧ (ψ ∧ τ) ∧ ch ⟶ ϑ
  using assms by auto



lemma MMI_3bitr4rd: assumes A1: φ ⟶
  ψ ⟷ ch and
   A2: φ ⟶
  ϑ ⟷ ψ and
   A3: φ ⟶
  τ ⟷ ch
  shows φ ⟶
  τ ⟷ ϑ
  using assms by auto

lemma MMI_3anrev:
  shows φ ∧ ψ ∧ ch ⟷ ch ∧ ψ ∧ φ
 by auto

lemma MMI_eqtr4: assumes A1: A = B and
   A2: C = B
  shows A = C
  using assms by auto

lemma MMI_anidm:
  shows φ ∧ φ ⟷ φ
 by auto

lemma MMI_bi2anan9r: assumes A1: φ ⟶
  ψ ⟷ ch and
   A2: ϑ ⟶
  τ ⟷ η
  shows ϑ ∧ φ ⟶
  ψ ∧ τ ⟷ ch ∧ η
```

using assms **by** auto

lemma MMI_3imtr3g: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**
A2: $\psi \longleftrightarrow \vartheta$ **and**
A3: ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\vartheta \longrightarrow \tau$
using assms **by** auto

lemma MMI_a3d: **assumes** A1: $\varphi \longrightarrow$
$\neg\psi \longrightarrow \neg$ch
shows $\varphi \longrightarrow$ ch $\longrightarrow \psi$
using assms **by** auto

lemma MMI_sylan9bbr: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\vartheta \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\vartheta \wedge \varphi \longrightarrow$
$\psi \longleftrightarrow \tau$
using assms **by** auto

lemma MMI_sylan9bb: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\vartheta \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\varphi \wedge \vartheta \longrightarrow$
$\psi \longleftrightarrow \tau$
using assms **by** auto

lemma MMI_3bitr3g: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\psi \longleftrightarrow \vartheta$ **and**
A3: ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \tau$
using assms **by** auto

lemma MMI_pm5_21:
shows $\neg\varphi \wedge \neg\psi \longrightarrow$
$\varphi \longleftrightarrow \psi$
by auto

lemma MMI_an6:
shows ($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta \wedge \tau \wedge \eta \longleftrightarrow$
($\varphi \wedge \vartheta$) $\wedge$ ($\psi \wedge \tau$) $\wedge$ ch $\wedge \eta$
by auto

lemma MMI_syl3anl1: **assumes** A1: $(\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$ **and**
A2: $\eta \longrightarrow \varphi$
shows $(\eta \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
using assms **by** auto

lemma MMI_imp4a: **assumes** A1: $\varphi \longrightarrow$
$\psi \longrightarrow$
$\mathrm{ch} \longrightarrow$
$\vartheta \longrightarrow \tau$
shows $\varphi \longrightarrow$
$\psi \longrightarrow$
$\mathrm{ch} \wedge \vartheta \longrightarrow \tau$
using assms **by** auto

lemma (**in** MMIsar0) MMI_breqan12rd: **assumes** A1: $\varphi \longrightarrow \mathrm{A} = \mathrm{B}$ **and**
A2: $\psi \longrightarrow \mathrm{C} = \mathrm{D}$
shows
$\psi \wedge \varphi \longrightarrow \mathrm{A} < \mathrm{C} \longleftrightarrow \mathrm{B} < \mathrm{D}$
$\psi \wedge \varphi \longrightarrow \mathrm{A} \leq \mathrm{C} \longleftrightarrow \mathrm{B} \leq \mathrm{D}$
using assms **by** auto

lemma (**in** MMIsar0) MMI_3brtr4d: **assumes** A1: $\varphi \longrightarrow \mathrm{A} < \mathrm{B}$ **and**
A2: $\varphi \longrightarrow \mathrm{C} = \mathrm{A}$ **and**
A3: $\varphi \longrightarrow \mathrm{D} = \mathrm{B}$
shows $\varphi \longrightarrow \mathrm{C} < \mathrm{D}$
using assms **by** auto

lemma MMI_adantrrr: **assumes** A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge \mathrm{ch} \wedge \tau \longrightarrow \vartheta$
using assms **by** auto

lemma MMI_adantrlr: **assumes** A1: $\varphi \wedge \psi \wedge \mathrm{ch} \longrightarrow \vartheta$
shows $\varphi \wedge (\psi \wedge \tau) \wedge \mathrm{ch} \longrightarrow \vartheta$
using assms **by** auto

lemma MMI_imdistani: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow \mathrm{ch}$
shows $\varphi \wedge \psi \longrightarrow \varphi \wedge \mathrm{ch}$
using assms **by** auto

lemma MMI_anabss3: **assumes** A1: $(\varphi \wedge \psi) \wedge \psi \longrightarrow \mathrm{ch}$
shows $\varphi \wedge \psi \longrightarrow \mathrm{ch}$
using assms **by** auto

lemma MMI_mp3anl2: **assumes** A1: ψ **and**
A2: $(\varphi \wedge \psi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$
shows $(\varphi \wedge \mathrm{ch}) \wedge \vartheta \longrightarrow \tau$

```
  using assms by auto


lemma MMI_mpanl2: assumes A1: ψ and
    A2: (φ ∧ ψ) ∧ ch ⟶ ϑ
  shows φ ∧ ch ⟶ ϑ
  using assms by auto

lemma MMI_mpancom: assumes A1: ψ ⟶ φ and
    A2: φ ∧ ψ ⟶ ch
  shows ψ ⟶ ch
  using assms by auto

lemma MMI_or12:
  shows φ ∨ ψ ∨ ch ⟷ ψ ∨ φ ∨ ch
 by auto

lemma MMI_rcla4ev: assumes A1: ∀x. x = A ⟶  φ(x) ⟷ ψ
  shows A ∈ B ∧ ψ ⟶ ( ∃x∈B. φ(x) )
  using assms by auto

lemma MMI_jctir: assumes A1: φ ⟶ ψ and
    A2: ch
  shows φ ⟶ ψ ∧ ch
  using assms by auto

lemma MMI_iffalse:
  shows ¬φ ⟶  if(φ, A, B) = B
 by auto

lemma MMI_iftrue:
  shows φ ⟶  if(φ, A, B) = A
 by auto

lemma MMI_pm2_61d2: assumes A1: φ ⟶
  ¬ψ ⟶ ch and
    A2: ψ ⟶ ch
  shows φ ⟶ ch
  using assms by auto

lemma MMI_pm2_61dan: assumes A1: φ ∧ ψ ⟶ ch and
    A2: φ ∧ ¬ψ ⟶ ch
  shows φ ⟶ ch
  using assms by auto

lemma MMI_orcanai: assumes A1: φ ⟶ ψ ∨ ch
  shows φ ∧ ¬ψ ⟶ ch
  using assms by auto
```

```
lemma MMI_ifcl:
   shows A ∈ C ∧ B ∈ C ⟶  if(φ, A, B) ∈ C
  by auto

lemma MMI_imim2i: assumes A1: φ ⟶ ψ
   shows (ch ⟶ φ) ⟶ ch ⟶ ψ
   using assms by auto

lemma MMI_com13: assumes A1: φ ⟶
   ψ ⟶ ch ⟶ ϑ
   shows ch ⟶
   ψ ⟶
   φ ⟶ ϑ
   using assms by auto

lemma MMI_rcla4v: assumes A1: ∀x. x = A ⟶  φ(x) ⟷ ψ
   shows A ∈ B ⟶  (∀x∈B. φ(x)) ⟶ ψ
   using assms by auto

lemma MMI_syl5d: assumes A1: φ ⟶
   ψ ⟶ ch ⟶ ϑ and
    A2: φ ⟶ τ ⟶ ch
   shows φ ⟶
   ψ ⟶
   τ ⟶ ϑ
   using assms by auto

lemma MMI_eqcoms: assumes A1: A = B ⟶ φ
   shows B = A ⟶ φ
   using assms by auto



lemma MMI_rgen: assumes A1: ∀x. x ∈ A ⟶ φ(x)
   shows ∀x∈A. φ(x)
   using assms by auto

lemma (in MMIsar0) MMI_reex:
   shows ℝ = ℝ
  by auto

lemma MMI_sstri: assumes A1: A ⊆B and
    A2: B ⊆C
   shows A ⊆C
   using assms by auto

lemma MMI_ssexi: assumes A1: B = B and
    A2: A ⊆B
```

```
   shows A = A
   using assms by auto
```

end

104 Complex numbers in Metamatah - introduction

theory MMI_Complex_ZF imports MMI_logic_and_sets

begin

This theory contains theorems (with proofs) about complex numbers imported from the Metamath's set.mm database. The original Metamath proofs were mostly written by Norman Megill, see the Metamath Proof Explorer pages for full atribution. This theory contains about 200 theorems from "recnt" to "div11t".

```
  lemma (in MMIsar0) MMI_recnt:
   shows A ∈ ℝ ⟶ A ∈ ℂ
proof -
   have S1: ℝ ⊆ ℂ by (rule MMI_axresscn)
   from S1 show A ∈ ℝ ⟶ A ∈ ℂ by (rule MMI_sseli)
qed

lemma (in MMIsar0) MMI_recn: assumes A1: A ∈ ℝ
   shows A ∈ ℂ
proof -
   have S1: ℝ ⊆ ℂ by (rule MMI_axresscn)
   from A1 have S2: A ∈ ℝ.
   from S1 S2 show A ∈ ℂ by (rule MMI_sselii)
qed

lemma (in MMIsar0) MMI_recnd: assumes A1: φ ⟶ A ∈ ℝ
   shows φ ⟶ A ∈ ℂ
proof -
   from A1 have S1: φ ⟶ A ∈ ℝ.
   have S2: A ∈ ℝ ⟶ A ∈ ℂ by (rule MMI_recnt)
   from S1 S2 show φ ⟶ A ∈ ℂ by (rule MMI_syl)
qed

lemma (in MMIsar0) MMI_elimne0:
   shows if ( A ≠ 0 , A , 1 ) ≠ 0
proof -
   have S1: A = if ( A ≠ 0 , A , 1 ) ⟶
     ( A ≠ 0 ⟷ if ( A ≠ 0 , A , 1 ) ≠ 0 ) by (rule MMI_neeq1)
   have S2: 1 = if ( A ≠ 0 , A , 1 ) ⟶
     ( 1 ≠ 0 ⟷ if ( A ≠ 0 , A , 1 ) ≠ 0 ) by (rule MMI_neeq1)
```

```
   have S3: 1 ≠ 0 by (rule MMI_ax1ne0)
   from S1 S2 S3 show if ( A ≠ 0 , A , 1 ) ≠ 0 by (rule MMI_elimhyp)
qed

lemma (in MMIsar0) MMI_addex:
   shows + isASet
proof -
   have S1: ℂ isASet by (rule MMI_axcnex)
   have S2: ℂ isASet by (rule MMI_axcnex)
   from S1 S2 have S3: ( ℂ × ℂ ) isASet by (rule MMI_xpex)
   have S4: + : ( ℂ × ℂ ) → ℂ by (rule MMI_axaddopr)
   have S5: ( ℂ × ℂ ) isASet ⟶
     ( + : ( ℂ × ℂ ) → ℂ ⟶ + isASet ) by (rule MMI_fex)
   from S3 S4 S5 show + isASet by (rule MMI_mp2)
qed

lemma (in MMIsar0) MMI_mulex:
   shows · isASet
proof -
   have S1: ℂ isASet by (rule MMI_axcnex)
   have S2: ℂ isASet by (rule MMI_axcnex)
   from S1 S2 have S3: ( ℂ × ℂ ) isASet by (rule MMI_xpex)
   have S4: · : ( ℂ × ℂ ) → ℂ by (rule MMI_axmulopr)
   have S5: ( ℂ × ℂ ) isASet ⟶
     ( · : ( ℂ × ℂ ) → ℂ ⟶ · isASet ) by (rule MMI_fex)
   from S3 S4 S5 show · isASet by (rule MMI_mp2)
qed

lemma (in MMIsar0) MMI_adddirt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
  ( ( A + B ) · C ) = ( ( A · C ) + ( B · C ) )
proof -
   have S1: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( C · ( A + B ) ) = ( ( C · A ) + ( C · B ) )
     by (rule MMI_axdistr)
   from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( C · ( A + B ) ) = ( ( C · A ) + ( C · B ) ) by (rule MMI_3coml)
   have S3: ( ( A + B ) ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) · C ) = ( C · ( A + B ) ) by (rule MMI_axmulcom)
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) ∈ ℂ by (rule MMI_axaddcl)
   from S3 S4 have S5: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) · C ) = ( C · ( A + B ) ) by (rule MMI_sylan)
   from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) · C ) = ( C · ( A + B ) ) by (rule MMI_3impa)
   have S7: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · C ) = ( C · A )
     by (rule MMI_axmulcom)
   from S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · C ) = ( C ·
A )
     by (rule MMI_3adant2)
```

```
   have S9: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B · C ) = ( C · B )
     by (rule MMI_axmulcom)
   from S9 have S10: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B · C ) = ( C ·
B )
     by (rule MMI_3adant1)
   from S8 S10 have S11: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A · C ) + ( B · C ) ) = ( ( C · A ) + ( C · B ) )
     by (rule MMI_opreq12d)
   from S2 S6 S11 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) · C ) = ( ( A · C ) + ( B · C ) )
     by (rule MMI_3eqtr4d)
qed

lemma (in MMIsar0) MMI_addcl: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( A + B ) ∈ ℂ
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) ∈ ℂ by (rule MMI_axaddcl)
   from S1 S2 S3 show ( A + B ) ∈ ℂ by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_mulcl: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( A · B ) ∈ ℂ
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) ∈ ℂ by (rule MMI_axmulcl)
   from S1 S2 S3 show ( A · B ) ∈ ℂ by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_addcom: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( A + B ) = ( B + A )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) = ( B + A )
     by (rule MMI_axaddcom)
   from S1 S2 S3 show ( A + B ) = ( B + A ) by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_mulcom: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( A · B ) = ( B · A )
proof -
   from A1 have S1: A ∈ ℂ.
```

```
    from A2 have S2: B ∈ ℂ.
    have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) = ( B · A )
      by (rule MMI_axmulcom)
    from S1 S2 S3 show ( A · B ) = ( B · A ) by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_addass: assumes A1: A ∈ ℂ and
     A2: B ∈ ℂ and
     A3: C ∈ ℂ
    shows ( ( A + B ) + C ) = ( A + ( B + C ) )
proof -
    from A1 have S1: A ∈ ℂ.
    from A2 have S2: B ∈ ℂ.
    from A3 have S3: C ∈ ℂ.
    have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
      ( A + ( B + C ) ) by (rule MMI_axaddass)
    from S1 S2 S3 S4 show ( ( A + B ) + C ) =
      ( A + ( B + C ) ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_mulass: assumes A1: A ∈ ℂ and
     A2: B ∈ ℂ and
     A3: C ∈ ℂ
    shows ( ( A · B ) · C ) = ( A · ( B · C ) )
proof -
    from A1 have S1: A ∈ ℂ.
    from A2 have S2: B ∈ ℂ.
    from A3 have S3: C ∈ ℂ.
    have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A · B ) · C ) =
      ( A · ( B · C ) ) by (rule MMI_axmulass)
    from S1 S2 S3 S4 show ( ( A · B ) · C ) = ( A · ( B · C ) )
      by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_adddi: assumes A1: A ∈ ℂ and
     A2: B ∈ ℂ and
     A3: C ∈ ℂ
    shows ( A · ( B + C ) ) = ( ( A · B ) + ( A · C ) )
proof -
    from A1 have S1: A ∈ ℂ.
    from A2 have S2: B ∈ ℂ.
    from A3 have S3: C ∈ ℂ.
    have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · ( B + C ) ) =
      ( ( A · B ) + ( A · C ) ) by (rule MMI_axdistr)
    from S1 S2 S3 S4 show ( A · ( B + C ) ) =
      ( ( A · B ) + ( A · C ) ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_adddir: assumes A1: A ∈ ℂ and
```

```
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows ( ( A + B ) · C ) = ( ( A · C ) + ( B · C ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) · C ) =
     ( ( A · C ) + ( B · C ) ) by (rule MMI_adddirt)
   from S1 S2 S3 S4 show ( ( A + B ) · C ) =
     ( ( A · C ) + ( B · C ) ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_1cn:
   shows 1 ∈ ℂ
proof -
   have S1: 1 ∈ ℝ by (rule MMI_ax1re)
   from S1 show 1 ∈ ℂ by (rule MMI_recn)
qed

lemma (in MMIsar0) MMI_0cn:
   shows 0 ∈ ℂ
proof -
   have S1: ( ( i · i ) + 1 ) = 0 by (rule MMI_axi2m1)
   have S2: i ∈ ℂ by (rule MMI_axicn)
   have S3: i ∈ ℂ by (rule MMI_axicn)
   from S2 S3 have S4: ( i · i ) ∈ ℂ by (rule MMI_mulcl)
   have S5: 1 ∈ ℂ by (rule MMI_1cn)
   from S4 S5 have S6: ( ( i · i ) + 1 ) ∈ ℂ by (rule MMI_addcl)
   from S1 S6 show 0 ∈ ℂ by (rule MMI_eqeltrr)
qed

lemma (in MMIsar0) MMI_addid1: assumes A1: A ∈ ℂ
   shows ( A + 0 ) = A
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: A ∈ ℂ ⟶ ( A + 0 ) = A by (rule MMI_ax0id)
   from S1 S2 show ( A + 0 ) = A by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_addid2: assumes A1: A ∈ ℂ
   shows ( 0 + A ) = A
proof -
   have S1: 0 ∈ ℂ by (rule MMI_0cn)
   from A1 have S2: A ∈ ℂ.
   from S1 S2 have S3: ( 0 + A ) = ( A + 0 ) by (rule MMI_addcom)
   from A1 have S4: A ∈ ℂ.
   from S4 have S5: ( A + 0 ) = A by (rule MMI_addid1)
   from S3 S5 show ( 0 + A ) = A by (rule MMI_eqtr)
```

```
qed

lemma (in MMIsar0) MMI_mulid1: assumes A1: A ∈ ℂ
   shows ( A · 1 ) = A
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: A ∈ ℂ ⟶ ( A · 1 ) = A by (rule MMI_ax1id)
   from S1 S2 show ( A · 1 ) = A by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_mulid2: assumes A1: A ∈ ℂ
   shows ( 1 · A ) = A
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S2: A ∈ ℂ.
   from S1 S2 have S3: ( 1 · A ) = ( A · 1 ) by (rule MMI_mulcom)
   from A1 have S4: A ∈ ℂ.
   from S4 have S5: ( A · 1 ) = A by (rule MMI_mulid1)
   from S3 S5 show ( 1 · A ) = A by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_negex: assumes A1: A ∈ ℂ
   shows ∃ x ∈ ℂ . ( A + x ) = 0
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: A ∈ ℂ ⟶ ( ∃ x ∈ ℂ . ( A + x ) = 0 ) by (rule MMI_axnegex)
   from S1 S2 show ∃ x ∈ ℂ . ( A + x ) = 0 by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_recex: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ∃ x ∈ ℂ . ( A · x ) = 1
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: A ≠ 0.
   have S3: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( ∃ x ∈ ℂ . ( A · x ) = 1 )
     by (rule MMI_axrecex)
   from S1 S2 S3 show ∃ x ∈ ℂ . ( A · x ) = 1 by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_readdcl: assumes A1: A ∈ ℝ and
    A2: B ∈ ℝ
   shows ( A + B ) ∈ ℝ
proof -
   from A1 have S1: A ∈ ℝ.
   from A2 have S2: B ∈ ℝ.
```

```
   have S3: ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A + B ) ∈ ℝ by (rule MMI_axaddrcl)
   from S1 S2 S3 show ( A + B ) ∈ ℝ by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_remulcl: assumes A1: A ∈ ℝ and
    A2: B ∈ ℝ
   shows ( A · B ) ∈ ℝ
proof -
   from A1 have S1: A ∈ ℝ.
   from A2 have S2: B ∈ ℝ.
   have S3: ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A · B ) ∈ ℝ by (rule MMI_axmulrcl)
   from S1 S2 S3 show ( A · B ) ∈ ℝ by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_addcan: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows ( A + B ) = ( A + C ) ⟷ B = C
proof -
   from A1 have S1: A ∈ ℂ.
   from S1 have S2: ∃ x ∈ ℂ . ( A + x ) = 0 by (rule MMI_negex)
   from A1 have S3: A ∈ ℂ.
   from A2 have S4: B ∈ ℂ.
   { fix x
     have S5: ( x ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( x + A ) + B ) =
       ( x + ( A + B ) ) by (rule MMI_axaddass)
     from S4 S5 have S6: ( x ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( ( x + A ) + B ) =
       ( x + ( A + B ) ) by (rule MMI_mp3an3)
     from A3 have S7: C ∈ ℂ.
     have S8: ( x ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( x + A ) + C ) =
       ( x + ( A + C ) ) by (rule MMI_axaddass)
     from S7 S8 have S9: ( x ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( ( x + A ) + C ) =
       ( x + ( A + C ) ) by (rule MMI_mp3an3)
     from S6 S9 have S10: ( x ∈ ℂ ∧ A ∈ ℂ ) ⟶
       ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷
       ( x + ( A + B ) ) = ( x + ( A + C ) ) )
       by (rule MMI_eqeq12d)
     from S3 S10 have S11: x ∈ ℂ ⟶ ( ( ( x + A ) + B ) =
       ( ( x + A ) + C ) ⟷ ( x + ( A + B ) ) =
       ( x + ( A + C ) ) ) by (rule MMI_mpan2)
     have S12: ( A + B ) = ( A + C ) ⟶ ( x + ( A + B ) ) =
       ( x + ( A + C ) ) by (rule MMI_opreq2)
     from S11 S12 have S13: x ∈ ℂ ⟶ ( ( A + B ) = ( A + C ) ⟶
       ( ( x + A ) + B ) = ( ( x + A ) + C ) )
       by (rule MMI_syl5bir)
     from S13 have S14: ( x ∈ ℂ ∧ ( A + x ) = 0 ) ⟶ ( ( A + B ) =

       ( A + C ) ⟶ ( ( x + A ) + B ) =
       ( ( x + A ) + C ) ) by (rule MMI_adantr)
```

```
    from A1 have S15: A ∈ ℂ.
    have S16: ( A ∈ ℂ ∧ x ∈ ℂ ) ⟶ ( A + x ) = ( x + A )
      by (rule MMI_axaddcom)
    from S15 S16 have S17: x ∈ ℂ ⟶ ( A + x ) = ( x + A )
      by (rule MMI_mpan)
    from S17 have S18: x ∈ ℂ ⟶ ( ( A + x ) = 0 ⟷
      ( x + A ) = 0 ) by (rule MMI_eqeq1d)
    have S19: ( x + A ) = 0 ⟶ ( ( x + A ) + B ) =
      ( 0 + B ) by (rule MMI_opreq1)
    from A2 have S20: B ∈ ℂ.
    from S20 have S21: ( 0 + B ) = B by (rule MMI_addid2)
    from S19 S21 have S22: ( x + A ) = 0 ⟶
      ( ( x + A ) + B ) = B by (rule MMI_syl6eq)
    have S23: ( x + A ) = 0 ⟶ ( ( x + A ) + C ) =
      ( 0 + C ) by (rule MMI_opreq1)
    from A3 have S24: C ∈ ℂ.
    from S24 have S25: ( 0 + C ) = C by (rule MMI_addid2)
    from S23 S25 have S26: ( x + A ) = 0 ⟶
      ( ( x + A ) + C ) = C by (rule MMI_syl6eq)
    from S22 S26 have S27: ( x + A ) = 0 ⟶
      ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷ B = C )
      by (rule MMI_eqeq12d)
    from S18 S27 have S28: x ∈ ℂ ⟶ ( ( A + x ) = 0 ⟶
      ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷ B = C ) )
      by (rule MMI_syl6bi)
    from S28 have S29: ( x ∈ ℂ ∧ ( A + x ) = 0 ) ⟶
      ( ( ( x + A ) + B ) = ( ( x + A ) + C ) ⟷ B = C )
      by (rule MMI_imp)
    from S14 S29 have S30: ( x ∈ ℂ ∧ ( A + x ) = 0 ) ⟶
      ( ( A + B ) = ( A + C ) ⟶ B = C ) by (rule MMI_sylibd)
    from S30 have x ∈ ℂ ⟶ ( ( A + x ) = 0 ⟶
      ( ( A + B ) = ( A + C ) ⟶ B = C ) ) by (rule MMI_ex)
  } then have S31: ∀ x. (x ∈ ℂ ⟶ ( ( A + x ) = 0 ⟶
      ( ( A + B ) = ( A + C ) ⟶ B = C ) )) by auto
  from S31 have S32: ( ∃ x ∈ ℂ . ( A + x ) = 0 ) ⟶
    ( ( A + B ) = ( A + C ) ⟶ B = C ) by (rule MMI_r19_23aiv)
  from S2 S32 have S33: ( A + B ) = ( A + C ) ⟶ B = C
    by (rule MMI_ax_mp)
  have S34: B = C ⟶ ( A + B ) = ( A + C ) by (rule MMI_opreq2)
  from S33 S34 show ( A + B ) = ( A + C ) ⟷ B = C
    by (rule MMI_impbi)
qed

lemma (in MMIsar0) MMI_addcan2: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
  shows ( A + C ) = ( B + C ) ⟷ A = B
proof -
  from A1 have S1: A ∈ ℂ.
```

```
   from A3 have S2: C ∈ ℂ.
   from S1 S2 have S3: ( A + C ) = ( C + A ) by (rule MMI_addcom)
   from A2 have S4: B ∈ ℂ.
   from A3 have S5: C ∈ ℂ.
   from S4 S5 have S6: ( B + C ) = ( C + B ) by (rule MMI_addcom)
   from S3 S6 have S7: ( A + C ) = ( B + C ) ⟷
     ( C + A ) = ( C + B ) by (rule MMI_eqeq12i)
   from A3 have S8: C ∈ ℂ.
   from A1 have S9: A ∈ ℂ.
   from A2 have S10: B ∈ ℂ.
   from S8 S9 S10 have S11: ( C + A ) = ( C + B ) ⟷ A = B
     by (rule MMI_addcan)
   from S7 S11 show ( A + C ) = ( B + C ) ⟷ A = B by (rule MMI_bitr)
qed

lemma (in MMIsar0) MMI_addcant:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
  ( ( A + B ) = ( A + C ) ⟷ B = C )
proof -
   have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A + B ) = ( if ( A ∈ ℂ , A
, 0 ) + B ) by (rule MMI_opreq1)
   have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( A + C ) = ( if ( A ∈ ℂ , A , 0 ) + C ) by (rule MMI_opreq1)
   from S1 S2 have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( ( A + B ) = ( A + C ) ⟷
     ( if ( A ∈ ℂ , A , 0 ) + B ) = ( if ( A ∈ ℂ , A , 0 ) + C ) )
     by (rule MMI_eqeq12d)
   from S3 have S4: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( ( ( A + B ) = ( A + C ) ⟷ B = C ) ⟷
     ( ( if ( A ∈ ℂ , A , 0 ) + B ) = ( if ( A ∈ ℂ , A , 0 ) + C )
     ⟷ B = C ) ) by (rule MMI_bibi1d)
   have S5: B = if ( B ∈ ℂ , B , 0 ) ⟶
     ( if ( A ∈ ℂ , A , 0 ) + B ) =
     ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
   from S5 have S6: B = if ( B ∈ ℂ , B , 0 ) ⟶
     ( ( if ( A ∈ ℂ , A , 0 ) + B ) = ( if ( A ∈ ℂ , A , 0 ) + C )
     ⟷ ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
     ( if ( A ∈ ℂ , A , 0 ) + C ) ) by (rule MMI_eqeq1d)
   have S7: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( B = C ⟷
     if ( B ∈ ℂ , B , 0 ) = C ) by (rule MMI_eqeq1)
   from S6 S7 have S8: B = if ( B ∈ ℂ , B , 0 ) ⟶
     ( ( ( if ( A ∈ ℂ , A , 0 ) + B ) =
     ( if ( A ∈ ℂ , A , 0 ) + C ) ⟷ B = C ) ⟷
     ( ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
     ( if ( A ∈ ℂ , A , 0 ) + C ) ⟷ if ( B ∈ ℂ , B , 0 ) = C ) )
     by (rule MMI_bibi12d)
   have S9: C = if ( C ∈ ℂ , C , 0 ) ⟶ ( if ( A ∈ ℂ , A , 0 ) + C
) =
     ( if ( A ∈ ℂ , A , 0 ) + if ( C ∈ ℂ , C , 0 ) )
```

```
    by (rule MMI_opreq2)
  from S9 have S10: C = if ( C ∈ ℂ , C , 0 ) ⟶
    ( ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
    ( if ( A ∈ ℂ , A , 0 ) + C ) ⟷
    ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
    ( if ( A ∈ ℂ , A , 0 ) + if ( C ∈ ℂ , C , 0 ) ) )
    by (rule MMI_eqeq2d)
  have S11: C = if ( C ∈ ℂ , C , 0 ) ⟶ ( if ( B ∈ ℂ , B , 0 ) = C
⟷
    if ( B ∈ ℂ , B , 0 ) = if ( C ∈ ℂ , C , 0 ) ) by (rule MMI_eqeq2)
  from S10 S11 have S12: C = if ( C ∈ ℂ , C , 0 ) ⟶
    ( ( ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
    ( if ( A ∈ ℂ , A , 0 ) + C ) ⟷ if ( B ∈ ℂ , B , 0 ) = C ) ⟷

    ( ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
    ( if ( A ∈ ℂ , A , 0 ) + if ( C ∈ ℂ , C , 0 ) ) ⟷
    if ( B ∈ ℂ , B , 0 ) = if ( C ∈ ℂ , C , 0 ) ) ) by (rule MMI_bibi12d)
  have S13: 0 ∈ ℂ by (rule MMI_0cn)
  from S13 have S14: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
  have S15: 0 ∈ ℂ by (rule MMI_0cn)
  from S15 have S16: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
  have S17: 0 ∈ ℂ by (rule MMI_0cn)
  from S17 have S18: if ( C ∈ ℂ , C , 0 ) ∈ ℂ by (rule MMI_elimel)
  from S14 S16 S18 have S19:
    ( if ( A ∈ ℂ , A , 0 ) + if ( B ∈ ℂ , B , 0 ) ) =
    ( if ( A ∈ ℂ , A , 0 ) + if ( C ∈ ℂ , C , 0 ) ) ⟷
    if ( B ∈ ℂ , B , 0 ) = if ( C ∈ ℂ , C , 0 ) by (rule MMI_addcan)
  from S4 S8 S12 S19 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) = ( A + C ) ⟷ B = C ) by (rule MMI_dedth3h)
qed

lemma (in MMIsar0) MMI_addcan2t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + C ) = ( B + C ) ⟷

 A = B )
proof -
  have S1: ( C ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( C + A ) = ( A + C )
    by (rule MMI_axaddcom)
  from S1 have S2: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( C + A ) =
    ( A + C ) by (rule MMI_3adant3)
  have S3: ( C ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( C + B ) = ( B + C )
    by (rule MMI_axaddcom)
  from S3 have S4: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( C + B ) =
    ( B + C ) by (rule MMI_3adant2)
  from S2 S4 have S5: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( C + A ) = ( C + B ) ⟷ ( A + C ) = ( B + C ) )
    by (rule MMI_eqeq12d)
  have S6: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( C + A ) =
    ( C + B ) ⟷ A = B ) by (rule MMI_addcant)
```

```
  from S5 S6 have S7: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + C ) =

    ( B + C ) ⟷ A = B ) by (rule MMI_bitr3d)
  from S7 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + C ) =
    ( B + C ) ⟷ A = B ) by (rule MMI_3coml)
qed
```

```
lemma (in MMIsar0) MMI_add12t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + ( B + C ) ) =
  ( B + ( A + C ) )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) = ( B + A )
    by (rule MMI_axaddcom)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
    ( ( B + A ) + C ) by (rule MMI_opreq1d)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) + C ) = ( ( B + A ) + C )
    by (rule MMI_3adant3)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
    ( A + ( B + C ) ) by (rule MMI_axaddass)
  have S5: ( B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( B + A ) + C ) =
    ( B + ( A + C ) ) by (rule MMI_axaddass)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( B + A ) + C ) = ( B + ( A + C ) ) by (rule MMI_3com12)
  from S3 S4 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( A + ( B + C ) ) = ( B + ( A + C ) )
    by (rule MMI_3eqtr3d)
qed
```

```
lemma (in MMIsar0) MMI_add23t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
  ( ( A + C ) + B )
proof -
  have S1: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B + C ) = ( C + B )
    by (rule MMI_axaddcom)
  from S1 have S2: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + ( B + C ) ) =
    ( A + ( C + B ) ) by (rule MMI_opreq2d)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( A + ( B + C ) ) = ( A + ( C + B ) )
    by (rule MMI_3adant1)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
    ( A + ( B + C ) ) by (rule MMI_axaddass)
  have S5: ( A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + C ) + B ) =
    ( A + ( C + B ) ) by (rule MMI_axaddass)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + C ) + B ) = ( A + ( C + B ) ) by (rule MMI_3com23)
  from S3 S4 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
```

```
    ( ( A + B ) + C ) = ( ( A + C ) + B )
    by (rule MMI_3eqtr4d)
qed

lemma (in MMIsar0) MMI_add4t:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
  ( ( A + B ) + ( C + D ) ) = ( ( A + C ) + ( B + D ) )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) + C ) = ( ( A + C ) + B ) by (rule MMI_add23t)
   from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( ( A + B ) + C ) + D ) =
     ( ( ( A + C ) + B ) + D ) by (rule MMI_opreq1d)
   from S2 have S3: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
     ( ( ( A + B ) + C ) + D ) =
     ( ( ( A + C ) + B ) + D ) by (rule MMI_3expa)
   from S3 have S4: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( ( A + B ) + C ) + D ) =
     ( ( ( A + C ) + B ) + D ) by (rule MMI_adantrr)
   have S5: ( ( A + B ) ∈ ℂ ∧ C ∈ ℂ ∧ D ∈ ℂ ) ⟶
     ( ( ( A + B ) + C ) + D ) =
     ( ( A + B ) + ( C + D ) ) by (rule MMI_axaddass)
   from S5 have S6: ( ( A + B ) ∈ ℂ ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( ( A + B ) + C ) + D ) =
     ( ( A + B ) + ( C + D ) ) by (rule MMI_3expb)
   have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) ∈ ℂ by (rule MMI_axaddcl)
   from S6 S7 have S8: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( ( A + B ) + C ) + D ) =
     ( ( A + B ) + ( C + D ) ) by (rule MMI_sylan)
   have S9: ( ( A + C ) ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ ) ⟶
     ( ( ( A + C ) + B ) + D ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_axaddass)
   from S9 have S10: ( ( A + C ) ∈ ℂ ∧ ( B ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( ( A + C ) + B ) + D ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_3expb)
   have S11: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + C ) ∈ ℂ by (rule MMI_axaddcl)
   from S10 S11 have S12: ( ( A ∈ ℂ ∧ C ∈ ℂ ) ∧ ( B ∈ ℂ ∧ D ∈ ℂ )
) ⟶
     ( ( ( A + C ) + B ) + D ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_sylan)
   from S12 have S13: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( ( A + C ) + B ) + D ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_an4s)
   from S4 S8 S13 show ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( A + B ) + ( C + D ) ) =
```

```
    ( ( A + C ) + ( B + D ) ) by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_add42t:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
  ( ( A + B ) + ( C + D ) ) = ( ( A + C ) + ( D + B ) )
proof -
   have S1: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( A + B ) + ( C + D ) ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_add4t)
   have S2: ( B ∈ ℂ ∧ D ∈ ℂ ) ⟶ ( B + D ) =
     ( D + B ) by (rule MMI_axaddcom)
   from S2 have S3: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( B + D ) = ( D + B ) by (rule MMI_ad2ant2l)
   from S3 have S4: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( A + C ) + ( B + D ) ) =
     ( ( A + C ) + ( D + B ) ) by (rule MMI_opreq2d)
   from S1 S4 show ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( A + B ) + ( C + D ) ) =
     ( ( A + C ) + ( D + B ) ) by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_add12: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows ( A + ( B + C ) ) = ( B + ( A + C ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + ( B + C ) ) =
     ( B + ( A + C ) ) by (rule MMI_add12t)
   from S1 S2 S3 S4 show ( A + ( B + C ) ) =
     ( B + ( A + C ) ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_add23: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows ( ( A + B ) + C ) = ( ( A + C ) + B )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) + C ) = ( ( A + C ) + B ) by (rule MMI_add23t)
   from S1 S2 S3 S4 show ( ( A + B ) + C ) =
```

```
    ( ( A + C ) + B ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_add4: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ and
    A4: D ∈ ℂ
   shows ( ( A + B ) + ( C + D ) ) =
  ( ( A + C ) + ( B + D ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI_pm3_2i)
   from A3 have S4: C ∈ ℂ.
   from A4 have S5: D ∈ ℂ.
   from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI_pm3_2i)
   have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( A + B ) + ( C + D ) ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_add4t)
   from S3 S6 S7 show ( ( A + B ) + ( C + D ) ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_add42: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ and
    A4: D ∈ ℂ
   shows ( ( A + B ) + ( C + D ) ) =
  ( ( A + C ) + ( D + B ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   from A4 have S4: D ∈ ℂ.
   from S1 S2 S3 S4 have S5: ( ( A + B ) + ( C + D ) ) =
     ( ( A + C ) + ( B + D ) ) by (rule MMI_add4)
   from A2 have S6: B ∈ ℂ.
   from A4 have S7: D ∈ ℂ.
   from S6 S7 have S8: ( B + D ) = ( D + B ) by (rule MMI_addcom)
   from S8 have S9: ( ( A + C ) + ( B + D ) ) =
     ( ( A + C ) + ( D + B ) ) by (rule MMI_opreq2i)
   from S5 S9 show ( ( A + B ) + ( C + D ) ) =
     ( ( A + C ) + ( D + B ) ) by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_addid2t:
   shows A ∈ ℂ ⟶ ( 0 + A ) = A
proof -
   have S1: 0 ∈ ℂ by (rule MMI_0cn)
```

```
   have S2: ( 0 ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( 0 + A ) = ( A + 0 )
     by (rule MMI_axaddcom)
   from S1 S2 have S3: A ∈ ℂ ⟶ ( 0 + A ) = ( A + 0 )
     by (rule MMI_mpan)
   have S4: A ∈ ℂ ⟶ ( A + 0 ) = A by (rule MMI_ax0id)
   from S3 S4 show A ∈ ℂ ⟶ ( 0 + A ) = A by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_peano2cn:
   shows A ∈ ℂ ⟶ ( A + 1 ) ∈ ℂ
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   have S2: ( A ∈ ℂ ∧ 1 ∈ ℂ ) ⟶ ( A + 1 ) ∈ ℂ by (rule MMI_axaddcl)
   from S1 S2 show A ∈ ℂ ⟶ ( A + 1 ) ∈ ℂ by (rule MMI_mpan2)
qed



lemma (in MMIsar0) MMI_peano2re:
   shows A ∈ ℝ ⟶ ( A + 1 ) ∈ ℝ
proof -
   have S1: 1 ∈ ℝ by (rule MMI_ax1re)
   have S2: ( A ∈ ℝ ∧ 1 ∈ ℝ ) ⟶ ( A + 1 ) ∈ ℝ by (rule MMI_axaddrcl)
   from S1 S2 show A ∈ ℝ ⟶ ( A + 1 ) ∈ ℝ by (rule MMI_mpan2)
qed

lemma (in MMIsar0) MMI_negeu: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ∃! x . x ∈ ℂ ∧ ( A + x ) = B
proof -
   { fix x y
     have S1: x = y ⟶ ( A + x ) = ( A + y ) by (rule MMI_opreq2)
     from S1 have x = y ⟶ ( ( A + x ) = B ⟷ ( A + y ) = B )
       by (rule MMI_eqeq1d)
   } then have S2: ∀x y. x = y ⟶ ( ( A + x ) = B ⟷
       ( A + y ) = B ) by simp
   from S2 have S3: ( ∃! x . x ∈ ℂ ∧ ( A + x ) = B ) ⟷
     ( ( ∃ x ∈ ℂ . ( A + x ) = B ) ∧
     ( ∀ x ∈ ℂ . ∀ y ∈ ℂ . ( ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶
     x = y ) ) ) by (rule MMI_reu4)
   from A1 have S4: A ∈ ℂ.
   from S4 have S5: ∃ y ∈ ℂ . ( A + y ) = 0 by (rule MMI_negex)
   from A2 have S6: B ∈ ℂ.
   { fix y
     have S7: ( y ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( y + B ) ∈ ℂ by (rule MMI_axaddcl)
     from S6 S7 have S8: y ∈ ℂ ⟶ ( y + B ) ∈ ℂ by (rule MMI_mpan2)
     have S9: ( y + B ) ∈ ℂ ⟷ ( ∃ x ∈ ℂ . x = ( y + B ) )
       by (rule MMI_risset)
     from S8 S9 have S10: y ∈ ℂ ⟶ ( ∃ x ∈ ℂ . x = ( y + B ) )
```

```
      by (rule MMI_sylib)
   { fix x
      have S11: x = ( y + B ) ⟶ ( A + x ) =
( A + ( y + B ) ) by (rule MMI_opreq2)
      from A1 have S12: A ∈ ℂ.
      from A2 have S13: B ∈ ℂ.
      have S14: ( A ∈ ℂ ∧ y ∈ ℂ ∧ B ∈ ℂ ) ⟶
( ( A + y ) + B ) = ( A + ( y + B ) )
by (rule MMI_axaddass)
      from S12 S13 S14 have S15: y ∈ ℂ ⟶ ( ( A + y ) + B ) =
( A + ( y + B ) ) by (rule MMI_mp3an13)
      from S15 have S16: y ∈ ℂ ⟶ ( A + ( y + B ) ) =
( ( A + y ) + B ) by (rule MMI_eqcomd)
      from S11 S16 have S17: ( y ∈ ℂ ∧ x = ( y + B ) )
⟶ ( A + x ) = ( ( A + y ) + B ) by (rule MMI_sylan9eqr)
      have S18: ( A + y ) = 0 ⟶
( ( A + y ) + B ) = ( 0 + B ) by (rule MMI_opreq1)
      from A2 have S19: B ∈ ℂ.
      from S19 have S20: ( 0 + B ) = B by (rule MMI_addid2)
      from S18 S20 have S21: ( A + y ) = 0 ⟶
( ( A + y ) + B ) = B by (rule MMI_syl6eq)
      from S17 S21 have S22: ( ( A + y ) = 0 ∧ ( y ∈ ℂ ∧ x =
( y + B ) ) ) ⟶ ( A + x ) = B by (rule MMI_sylan9eqr)
      from S22 have S23: ( A + y ) = 0 ⟶
( y ∈ ℂ ⟶ ( x = ( y + B ) ⟶ ( A + x ) = B ) )
by (rule MMI_exp32)
      from S23 have S24: ( y ∈ ℂ ∧ ( A + y ) = 0 ) ⟶
( x = ( y + B ) ⟶ ( A + x ) = B ) by (rule MMI_impcom)
      from S24 have ( y ∈ ℂ ∧ ( A + y ) = 0 ) ⟶
( x ∈ ℂ ⟶ ( x = ( y + B ) ⟶ ( A + x ) = B ) )
by (rule MMI_a1d)
   } then have S25: ∀ x. ( y ∈ ℂ ∧ ( A + y ) = 0 ) ⟶
( x ∈ ℂ ⟶ ( x = ( y + B ) ⟶ ( A + x ) = B ) ) by auto
   from S25 have S26: ( y ∈ ℂ ∧ ( A + y ) = 0 ) ⟶
      ( ∀ x ∈ ℂ . ( x = ( y + B ) ⟶ ( A + x ) = B ) )
      by (rule MMI_r19_21aiv)
   from S26 have S27: y ∈ ℂ ⟶ ( ( A + y ) = 0 ⟶
      ( ∀ x ∈ ℂ . ( x = ( y + B ) ⟶ ( A + x ) = B ) ) )
      by (rule MMI_ex)
   have S28: ( ∀ x ∈ ℂ . ( x = ( y + B ) ⟶ ( A + x ) = B ) )
      ⟶ ( ( ∃ x ∈ ℂ . x = ( y + B ) ) ⟶
      ( ∃ x ∈ ℂ . ( A + x ) = B ) ) by (rule MMI_r19_22)
   from S27 S28 have S29: y ∈ ℂ ⟶ ( ( A + y ) = 0 ⟶
      ( ( ∃ x ∈ ℂ . x = ( y + B ) ) ⟶
      ( ∃ x ∈ ℂ . ( A + x ) = B ) ) ) by (rule MMI_syl6)
   from S10 S29 have y ∈ ℂ ⟶ ( ( A + y ) = 0 ⟶
      ( ∃ x ∈ ℂ . ( A + x ) = B ) ) by (rule MMI_mpid)
 } then have S30: ∀ y. y ∈ ℂ ⟶ ( ( A + y ) = 0 ⟶
      ( ∃ x ∈ ℂ . ( A + x ) = B ) ) by simp
```

```
   from S30 have S31: ( ∃ y ∈ ℂ . ( A + y ) = 0 ) ⟶
     ( ∃ x ∈ ℂ . ( A + x ) = B ) by (rule MMI_r19_23aiv)
   from S5 S31 have S32: ∃ x ∈ ℂ . ( A + x ) = B by (rule MMI_ax_mp)
   from A1 have S33: A ∈ ℂ.
   { fix x y
     have S34: ( A ∈ ℂ ∧ x ∈ ℂ ∧ y ∈ ℂ ) ⟶
       ( ( A + x ) = ( A + y ) ⟷ x = y ) by (rule MMI_addcant)
     have S35: ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶
       ( A + x ) = ( A + y ) by (rule MMI_eqtr3t)
     from S34 S35 have S36: ( A ∈ ℂ ∧ x ∈ ℂ ∧ y ∈ ℂ ) ⟶
       ( ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶ x = y )
       by (rule MMI_syl5bi)
     from S33 S36 have ( x ∈ ℂ ∧ y ∈ ℂ ) ⟶
       ( ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶ x = y )
       by (rule MMI_mp3an1)
   } then have S37: ∀x y . ( x ∈ ℂ ∧ y ∈ ℂ ) ⟶
       ( ( ( A + x ) = B ∧ ( A + y ) = B ) ⟶ x = y ) by auto
   from S37 have S38: ∀ x ∈ ℂ . ∀ y ∈ ℂ . ( ( ( A + x ) = B ∧
     ( A + y ) = B ) ⟶ x = y ) by (rule MMI_rgen2)
   from S3 S32 S38 show ∃! x . x ∈ ℂ ∧ ( A + x ) = B
     by (rule MMI_mpbir2an)
qed
```

```
lemma (in MMIsar0) MMI_subval: assumes A ∈ ℂ  B ∈ ℂ
  shows A - B = ⋃ { x ∈ ℂ . B + x = A }
  using sub_def by simp
```

```
lemma (in MMIsar0) MMI_df_neg: shows (- A) = 0 - A
  using cneg_def by simp
```

```
lemma (in MMIsar0) MMI_negeq:
   shows A = B ⟶ (-A) = (- B)
proof -
   have S1: A = B ⟶ ( 0 - A ) = ( 0 - B ) by (rule MMI_opreq2)
   have S2: (-A) = ( 0 - A ) by (rule MMI_df_neg)
   have S3: (-B) = ( 0 - B ) by (rule MMI_df_neg)
   from S1 S2 S3 show A = B ⟶ (-A) = (-B) by (rule MMI_3eqtr4g)
qed
```

```
lemma (in MMIsar0) MMI_negeqi: assumes A1: A = B
   shows (-  A) = (-B)
proof -
   from A1 have S1: A = B.
```

```
   have S2: A = B ⟶ (-A) = (-B) by (rule MMI_negeq)
   from S1 S2 show (-A) = (-B) by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_negeqd: assumes A1: φ ⟶ A = B
   shows φ ⟶ (-A) = (-B)
proof -
   from A1 have S1: φ ⟶ A = B.
   have S2: A = B ⟶ (-A) = (-B) by (rule MMI_negeq)
   from S1 S2 show φ ⟶ (-A) = (-B) by (rule MMI_syl)
qed



lemma (in MMIsar0) MMI_hbneg: assumes A1: y ∈ A ⟶ ( ∀ x . y ∈ A )

   shows y ∈ ((-  A)) ⟶ ( ∀ x . (y ∈ ((-  A)) ) )
  using assms by auto

lemma (in MMIsar0) MMI_minusex:
   shows ((-  A)) isASet by auto



lemma (in MMIsar0) MMI_subcl: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( A - B ) ∈ ℂ
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S1 S2 have S3: ( A - B ) = ⋃ { x ∈ ℂ . ( B + x ) = A }
     by (rule MMI_subval)
   from A2 have S4: B ∈ ℂ.
   from A1 have S5: A ∈ ℂ.
   from S4 S5 have S6: ∃! x . x ∈ ℂ ∧ ( B + x ) = A by (rule MMI_negeu)
   have S7: ( ∃! x . x ∈ ℂ ∧ ( B + x ) = A ) ⟶
    ⋃ { x ∈ ℂ . ( B + x ) = A } ∈ ℂ by (rule MMI_reucl)
   from S6 S7 have S8: ⋃ { x ∈ ℂ . ( B + x ) = A } ∈ ℂ
     by (rule MMI_ax_mp)
   from S3 S8 show ( A - B ) ∈ ℂ by simp
qed

lemma (in MMIsar0) MMI_subclt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ
proof -
   have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A - B ) =
     ( if ( A ∈ ℂ , A , 0 ) - B ) by (rule MMI_opreq1)
   from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( ( A - B ) ∈ ℂ ⟷
```

```
    ( if ( A ∈ ℂ , A , 0 ) - B ) ∈ ℂ ) by (rule MMI_eleq1d)
  have S3: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( if ( A ∈ ℂ , A , 0 ) - B
) =
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
  from S3 have S4: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( ( if ( A ∈ ℂ , A , 0 ) - B ) ∈ ℂ ⟷
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) ∈ ℂ )
    by (rule MMI_eleq1d)
  have S5: 0 ∈ ℂ by (rule MMI_0cn)
  from S5 have S6: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
  have S7: 0 ∈ ℂ by (rule MMI_0cn)
  from S7 have S8: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
  from S6 S8 have S9:
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) ∈ ℂ
    by (rule MMI_subcl)
  from S2 S4 S9 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ
    by (rule MMI_dedth2h)
qed

lemma (in MMIsar0) MMI_negclt:
  shows A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ
proof -
  have S1: 0 ∈ ℂ by (rule MMI_0cn)
  have S2: ( 0 ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( 0 - A ) ∈ ℂ by (rule MMI_subclt)
  from S1 S2 have S3: A ∈ ℂ ⟶ ( 0 - A ) ∈ ℂ by (rule MMI_mpan)
  have S4: ( (-  A) ) = ( 0 - A ) by (rule MMI_df_neg)
  from S3 S4 show A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI_syl5eqel)
qed

lemma (in MMIsar0) MMI_negcl: assumes A1: A ∈ ℂ
  shows ( (-  A) ) ∈ ℂ
proof -
  from A1 have S1: A ∈ ℂ.
  have S2: A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI_negclt)
  from S1 S2 show ( (-  A) ) ∈ ℂ by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_subadd: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ and
   A3: C ∈ ℂ
  shows ( A - B ) = C ⟷ ( B + C ) = A
proof -
  from A3 have S1: C ∈ ℂ.
  { fix x
    have S2: x = C ⟶ ( ( A - B ) = x ⟷ ( A - B ) = C )
      by (rule MMI_eqeq2)
    have S3: x = C ⟶ ( B + x ) = ( B + C ) by (rule MMI_opreq2)
    from S3 have S4: x = C ⟶ ( ( B + x ) = A ⟷ ( B + C ) = A )
      by (rule MMI_eqeq1d)
```

```
      from S2 S4 have x = C ⟶ ( ( ( A - B ) = x ⟷
        ( B + x ) = A ) ⟷ ( ( A - B ) = C ⟷ ( B + C ) = A ) )
        by (rule MMI_bibi12d)
    } then have S5: ∀x. x = C ⟶ ( ( ( A - B ) = x ⟷
        ( B + x ) = A ) ⟷ ( ( A - B ) = C ⟷
        ( B + C ) = A ) ) by simp
    from A2 have S6: B ∈ ℂ.
    from A1 have S7: A ∈ ℂ.
    from S6 S7 have S8: ∃! x . x ∈ ℂ ∧ ( B + x ) = A by (rule MMI_negeu)
    { fix x
      have S9: ( x ∈ ℂ ∧ ( ∃! x . x ∈ ℂ ∧ ( B + x ) = A ) ⟶
        ( ( B + x ) = A ) ⟷ ⋃ { x ∈ ℂ . ( B + x ) = A } = x )
        by (rule MMI_reuuni1)
      from S8 S9 have x ∈ ℂ ⟶ ( ( B + x ) = A ⟷
        ⋃ { x ∈ ℂ . ( B + x ) = A } = x ) by (rule MMI_mpan2)
    } then have S10: ∀ x. x ∈ ℂ ⟶ ( ( B + x ) = A ⟷
        ⋃ { x ∈ ℂ . ( B + x ) = A } = x ) by blast
    from A1 have S11: A ∈ ℂ.
    from A2 have S12: B ∈ ℂ.
    from S11 S12 have S13: ( A - B ) = ⋃ { x ∈ ℂ . ( B + x ) = A }
      by (rule MMI_subval)
    from S13 have S14: ∀x. ( A - B ) = x ⟷
      ⋃ { x ∈ ℂ . ( B + x ) = A } = x by simp
    from S10 S14 have S15: ∀x. x ∈ ℂ ⟶ ( ( A - B ) = x ⟷
      ( B + x ) = A ) by (rule MMI_syl6rbbr)
    from S5 S15 have S16: C ∈ ℂ ⟶ ( ( A - B ) = C ⟷
      ( B + C ) = A ) by (rule MMI_vtoclga)
    from S1 S16 show ( A - B ) = C ⟷ ( B + C ) = A
      by (rule MMI_ax_mp)
qed
```

```
lemma (in MMIsar0) MMI_subsub23: assumes A1: A ∈ ℂ and
     A2: B ∈ ℂ and
     A3: C ∈ ℂ
   shows ( A - B ) = C ⟷ ( A - C ) = B
proof -
   from A2 have S1: B ∈ ℂ.
   from A3 have S2: C ∈ ℂ.
   from S1 S2 have S3: ( B + C ) = ( C + B ) by (rule MMI_addcom)
   from S3 have S4: ( B + C ) = A ⟷ ( C + B ) = A
     by (rule MMI_eqeq1i)
   from A1 have S5: A ∈ ℂ.
   from A2 have S6: B ∈ ℂ.
   from A3 have S7: C ∈ ℂ.
   from S5 S6 S7 have S8: ( A - B ) = C ⟷ ( B + C ) = A
     by (rule MMI_subadd)
```

```
  from A1 have S9: A ∈ ℂ.
  from A3 have S10: C ∈ ℂ.
  from A2 have S11: B ∈ ℂ.
  from S9 S10 S11 have S12: ( A - C ) = B ⟷ ( C + B ) = A
    by (rule MMI_subadd)
  from S4 S8 S12 show ( A - B ) = C ⟷ ( A - C ) = B
    by (rule MMI_3bitr4)
qed

lemma (in MMIsar0) MMI_subaddt:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A - B ) = C ⟷
 ( B + C ) = A )
proof -
  have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A - B ) =
    ( if ( A ∈ ℂ , A , 0 ) - B ) by (rule MMI_opreq1)
  from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( ( A - B ) = C ⟷

    ( if ( A ∈ ℂ , A , 0 ) - B ) = C ) by (rule MMI_eqeq1d)
  have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( ( B + C ) = A ⟷
    ( B + C ) = if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_eqeq2)
  from S2 S3 have S4: A = if ( A ∈ ℂ , A , 0 ) ⟶
    ( ( ( A - B ) = C ⟷ ( B + C ) = A ) ⟷
    ( ( if ( A ∈ ℂ , A , 0 ) - B ) = C ⟷ ( B + C ) =
    if ( A ∈ ℂ , A , 0 ) ) ) by (rule MMI_bibi12d)
  have S5: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( if ( A ∈ ℂ , A , 0 ) - B ) =
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
  from S5 have S6: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( ( if ( A ∈ ℂ , A , 0 ) - B ) = C ⟷
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = C )
    by (rule MMI_eqeq1d)
  have S7: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( B + C ) =
    ( if ( B ∈ ℂ , B , 0 ) + C ) by (rule MMI_opreq1)
  from S7 have S8: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( ( B + C ) = if ( A ∈ ℂ , A , 0 ) ⟷
    ( if ( B ∈ ℂ , B , 0 ) + C ) = if ( A ∈ ℂ , A , 0 ) )
    by (rule MMI_eqeq1d)
  from S6 S8 have S9: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( ( ( if ( A ∈ ℂ , A , 0 ) - B ) = C ⟷
    ( B + C ) = if ( A ∈ ℂ , A , 0 ) ) ⟷
    ( ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = C ⟷
    ( if ( B ∈ ℂ , B , 0 ) + C ) = if ( A ∈ ℂ , A , 0 ) ) )
    by (rule MMI_bibi12d)
  have S10: C = if ( C ∈ ℂ , C , 0 ) ⟶
    ( ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = C ⟷
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) =
    if ( C ∈ ℂ , C , 0 ) ) by (rule MMI_eqeq2)
  have S11: C = if ( C ∈ ℂ , C , 0 ) ⟶
    ( if ( B ∈ ℂ , B , 0 ) + C ) =
```

```
      ( if ( B ∈ ℂ , B , 0 ) + if ( C ∈ ℂ , C , 0 ) ) by (rule MMI_opreq2)
    from S11 have S12: C = if ( C ∈ ℂ , C , 0 ) ⟶
      ( ( if ( B ∈ ℂ , B , 0 ) + C ) = if ( A ∈ ℂ , A , 0 ) ⟷
      ( if ( B ∈ ℂ , B , 0 ) + if ( C ∈ ℂ , C , 0 ) ) =
      if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_eqeq1d)
    from S10 S12 have S13: C = if ( C ∈ ℂ , C , 0 ) ⟶
      ( ( ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = C ⟷
      ( if ( B ∈ ℂ , B , 0 ) + C ) = if ( A ∈ ℂ , A , 0 ) ) ⟷
      ( ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) =
      if ( C ∈ ℂ , C , 0 ) ⟷
      ( if ( B ∈ ℂ , B , 0 ) + if ( C ∈ ℂ , C , 0 ) ) =
      if ( A ∈ ℂ , A , 0 ) ) ) by (rule MMI_bibi12d)
    have S14: 0 ∈ ℂ by (rule MMI_0cn)
    from S14 have S15: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
    have S16: 0 ∈ ℂ by (rule MMI_0cn)
    from S16 have S17: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
    have S18: 0 ∈ ℂ by (rule MMI_0cn)
    from S18 have S19: if ( C ∈ ℂ , C , 0 ) ∈ ℂ by (rule MMI_elimel)
    from S15 S17 S19 have S20:
      ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) =
      if ( C ∈ ℂ , C , 0 ) ⟷
      ( if ( B ∈ ℂ , B , 0 ) + if ( C ∈ ℂ , C , 0 ) ) =
      if ( A ∈ ℂ , A , 0 ) by (rule MMI_subadd)
    from S4 S9 S13 S20 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
      ( ( A - B ) = C ⟷ ( B + C ) = A ) by (rule MMI_dedth3h)
qed

lemma (in MMIsar0) MMI_pncan3t:
    shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( B - A ) ) = B
proof -
    have S1: ( B - A ) = ( B - A ) by (rule MMI_eqid)
    have S2: ( B ∈ ℂ ∧ A ∈ ℂ ∧ ( B - A ) ∈ ℂ ) ⟶
      ( ( B - A ) = ( B - A ) ⟷ ( A + ( B - A ) ) = B )
      by (rule MMI_subaddt)
    have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ B ∈ ℂ by (rule MMI_pm3_27)
    have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ A ∈ ℂ by (rule MMI_pm3_26)
    have S5: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( B - A ) ∈ ℂ by (rule MMI_subclt)
    from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( B - A ) ∈ ℂ
      by (rule MMI_ancoms)
    from S2 S3 S4 S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( B - A ) =
      ( B - A ) ⟷ ( A + ( B - A ) ) = B ) by (rule MMI_syl3anc)
    from S1 S7 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( B - A ) ) = B
      by (rule MMI_mpbii)
qed

lemma (in MMIsar0) MMI_pncan3: assumes A1: A ∈ ℂ and
     A2: B ∈ ℂ
    shows ( A + ( B - A ) ) = B
proof -
```

```
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( B - A ) ) = B
    by (rule MMI_pncan3t)
  from S1 S2 S3 show ( A + ( B - A ) ) = B by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_negidt:
  shows A ∈ ℂ ⟶ ( A + ( (- A) ) ) = 0
proof -
  have S1: 0 ∈ ℂ by (rule MMI_0cn)
  have S2: ( A ∈ ℂ ∧ 0 ∈ ℂ ) ⟶ ( A + ( 0 - A ) ) = 0
    by (rule MMI_pncan3t)
  from S1 S2 have S3: A ∈ ℂ ⟶ ( A + ( 0 - A ) ) = 0
    by (rule MMI_mpan2)
  have S4: ( (- A) ) = ( 0 - A ) by (rule MMI_df_neg)
  from S4 have S5: ( A + ( (- A) ) ) = ( A + ( 0 - A ) )
    by (rule MMI_opreq2i)
  from S3 S5 show A ∈ ℂ ⟶ ( A + ( (- A) ) ) = 0 by (rule MMI_syl5eq)
qed

lemma (in MMIsar0) MMI_negid: assumes A1: A ∈ ℂ
  shows ( A + ( (- A) ) ) = 0
proof -
  from A1 have S1: A ∈ ℂ.
  have S2: A ∈ ℂ ⟶ ( A + ( (- A) ) ) = 0 by (rule MMI_negidt)
  from S1 S2 show ( A + ( (- A) ) ) = 0 by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_negsub: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ
  shows ( A + ( (- B) ) ) = ( A - B )
proof -
  from A2 have S1: B ∈ ℂ.
  from A1 have S2: A ∈ ℂ.
  from A2 have S3: B ∈ ℂ.
  from S3 have S4: ( (- B) ) ∈ ℂ by (rule MMI_negcl)
  from S2 S4 have S5: ( A + ( (- B) ) ) ∈ ℂ by (rule MMI_addcl)
  from S1 S5 have S6: ( B + ( A + ( (- B) ) ) ) =
    ( ( A + ( (- B) ) ) + B ) by (rule MMI_addcom)
  from A1 have S7: A ∈ ℂ.
  from S4 have S8: ( (- B) ) ∈ ℂ .
  from A2 have S9: B ∈ ℂ.
  from S7 S8 S9 have S10: ( ( A + ( (- B) ) ) + B ) =
    ( A + ( ( (- B) ) + B ) ) by (rule MMI_addass)
  from S4 have S11: ( (- B) ) ∈ ℂ .
  from A2 have S12: B ∈ ℂ.
  from S11 S12 have S13: ( ( (- B) ) + B ) = ( B + ( (- B) ) )
    by (rule MMI_addcom)
```

```
   from A2 have S14: B ∈ ℂ.
   from S14 have S15: ( B + ( (-  B) ) ) = 0 by (rule MMI_negid)
   from S13 S15 have S16: ( ( (-  B) ) + B ) = 0 by (rule MMI_eqtr)
   from S16 have S17: ( A + ( ( (-  B) ) + B ) ) = ( A + 0 )
     by (rule MMI_opreq2i)
   from A1 have S18: A ∈ ℂ.
   from S18 have S19: ( A + 0 ) = A by (rule MMI_addid1)
   from S10 S17 S19 have S20: ( ( A + ( (-  B) ) ) + B ) = A
     by (rule MMI_3eqtr)
   from S6 S20 have S21: ( B + ( A + ( (-  B) ) ) ) = A
     by (rule MMI_eqtr)
   from A1 have S22: A ∈ ℂ.
   from A2 have S23: B ∈ ℂ.
   from S5 have S24: ( A + ( (-  B) ) ) ∈ ℂ .
   from S22 S23 S24 have S25: ( A - B ) = ( A + ( (-  B) ) ) ⟷
     ( B + ( A + ( (-  B) ) ) ) = A by (rule MMI_subadd)
   from S21 S25 have S26: ( A - B ) = ( A + ( (-  B) ) )
     by (rule MMI_mpbir)
   from S26 show ( A + ( (-  B) ) ) = ( A - B ) by (rule MMI_eqcomi)
qed

lemma (in MMIsar0) MMI_negsubt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( (-  B) ) ) = ( A - B )
proof -
   have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A + ( (-  B) ) ) =
     ( if ( A ∈ ℂ , A , 0 ) + ( (-  B) ) ) by (rule MMI_opreq1)
   have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A - B ) =
     ( if ( A ∈ ℂ , A , 0 ) - B ) by (rule MMI_opreq1)
   from S1 S2 have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( ( A + ( (-  B) ) ) = ( A - B ) ⟷
     ( if ( A ∈ ℂ , A , 0 ) + ( (-  B) ) ) =
     ( if ( A ∈ ℂ , A , 0 ) - B ) ) by (rule MMI_eqeq12d)
   have S4: B = if ( B ∈ ℂ , B , 0 ) ⟶
     ( (-  B) ) = ( - if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_negeq)
   from S4 have S5: B = if ( B ∈ ℂ , B , 0 ) ⟶
     ( if ( A ∈ ℂ , A , 0 ) + ( (-  B) ) ) =
     ( if ( A ∈ ℂ , A , 0 ) + ( - if ( B ∈ ℂ , B , 0 ) ) )
     by (rule MMI_opreq2d)
   have S6: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( if ( A ∈ ℂ , A , 0 ) - B
) =
     ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) )
     by (rule MMI_opreq2)
   from S5 S6 have S7: B = if ( B ∈ ℂ , B , 0 ) ⟶
     ( ( if ( A ∈ ℂ , A , 0 ) + ( (-  B) ) ) =
     ( if ( A ∈ ℂ , A , 0 ) - B ) ⟷
     ( if ( A ∈ ℂ , A , 0 ) + ( - if ( B ∈ ℂ , B , 0 ) ) ) =
     ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) )
     by (rule MMI_eqeq12d)
   have S8: 0 ∈ ℂ by (rule MMI_0cn)
```

```
  from S8 have S9: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
  have S10: 0 ∈ ℂ by (rule MMI_0cn)
  from S10 have S11: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
  from S9 S11 have S12:
    ( if ( A ∈ ℂ , A , 0 ) + ( - if ( B ∈ ℂ , B , 0 ) ) ) =
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) )
    by (rule MMI_negsub)
  from S3 S7 S12 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( (-  B) ) ) =
    ( A - B ) by (rule MMI_dedth2h)
qed

lemma (in MMIsar0) MMI_addsubasst:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
 ( A + ( B - C ) )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( - C ) ∈ ℂ ) ⟶
    ( ( A + B ) + ( - C ) ) =
    ( A + ( B + ( - C ) ) ) by (rule MMI_axaddass)
  have S2: C ∈ ℂ ⟶ ( - C ) ∈ ℂ by (rule MMI_negclt)
  from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) + ( - C ) ) =
    ( A + ( B + ( - C ) ) ) by (rule MMI_syl3an3)
  have S4: ( ( A + B ) ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) + ( - C ) ) = ( ( A + B ) - C )
    by (rule MMI_negsubt)
  have S5: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) ∈ ℂ by (rule MMI_axaddcl)
  from S4 S5 have S6: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) + ( - C ) ) = ( ( A + B ) - C )
    by (rule MMI_sylan)
  from S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) + ( - C ) ) = ( ( A + B ) - C )
    by (rule MMI_3impa)
  have S8: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B + ( - C ) ) = ( B - C )
    by (rule MMI_negsubt)
  from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( B + ( - C ) ) = ( B - C ) by (rule MMI_3adant1)
  from S9 have S10: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( A + ( B + ( - C ) ) ) = ( A + ( B - C ) )
    by (rule MMI_opreq2d)
  from S3 S7 S10 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) - C ) = ( A + ( B - C ) )
    by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_addsubt:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
 ( ( A - C ) + B )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) = ( B + A )
```

```
    by (rule MMI_axaddcom)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
    ( ( B + A ) - C ) by (rule MMI_opreq1d)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) - C ) = ( ( B + A ) - C )
    by (rule MMI_3adant3)
  have S4: ( B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( B + A ) - C ) =
    ( B + ( A - C ) ) by (rule MMI_addsubasst)
  from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( B + A ) - C ) = ( B + ( A - C ) ) by (rule MMI_3com12)
  have S6: ( B ∈ ℂ ∧ ( A - C ) ∈ ℂ ) ⟶ ( B + ( A - C ) ) =
    ( ( A - C ) + B ) by (rule MMI_axaddcom)
  from S6 have S7: B ∈ ℂ ⟶ ( ( A - C ) ∈ ℂ ⟶
    ( B + ( A - C ) ) = ( ( A - C ) + B ) ) by (rule MMI_ex)
  have S8: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A - C ) ∈ ℂ by (rule MMI_subclt)
  from S7 S8 have S9: B ∈ ℂ ⟶ ( ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( B + ( A - C ) ) = ( ( A - C ) + B ) ) by (rule MMI_syl5)
  from S9 have S10: B ∈ ℂ ⟶ ( A ∈ ℂ ⟶ ( C ∈ ℂ ⟶
    ( B + ( A - C ) ) = ( ( A - C ) + B ) ) )
    by (rule MMI_exp3a)
  from S10 have S11: A ∈ ℂ ⟶ ( B ∈ ℂ ⟶ ( C ∈ ℂ ⟶
    ( B + ( A - C ) ) = ( ( A - C ) + B ) ) )
    by (rule MMI_com12)
  from S11 have S12: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( B + ( A - C ) ) = ( ( A - C ) + B ) by (rule MMI_3imp)
  from S3 S5 S12 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) - C ) = ( ( A - C ) + B ) by (rule MMI_3eqtrd)
qed
```

```
lemma (in MMIsar0) MMI_addsub12t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + ( B - C ) ) =
 ( B + ( A - C ) )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) = ( B + A )
    by (rule MMI_axaddcom)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
    ( ( B + A ) - C ) by (rule MMI_opreq1d)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + B ) - C ) = ( ( B + A ) - C )
    by (rule MMI_3adant3)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
    ( A + ( B - C ) ) by (rule MMI_addsubasst)
  have S5: ( B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( B + A ) - C ) =
    ( B + ( A - C ) ) by (rule MMI_addsubasst)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( B + A ) - C ) = ( B + ( A - C ) ) by (rule MMI_3com12)
  from S3 S4 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
```

```
     ( A + ( B - C ) ) = ( B + ( A - C ) )
     by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_addsubass: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows ( ( A + B ) - C ) = ( A + ( B - C ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
     ( A + ( B - C ) ) by (rule MMI_addsubasst)
   from S1 S2 S3 S4 show ( ( A + B ) - C ) =
     ( A + ( B - C ) ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_addsub: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows ( ( A + B ) - C ) = ( ( A - C ) + B )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) - C ) =
     ( ( A - C ) + B ) by (rule MMI_addsubt)
   from S1 S2 S3 S4 show ( ( A + B ) - C ) =
     ( ( A - C ) + B ) by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_2addsubt:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
  ( ( ( A + B ) + C ) - D ) = ( ( ( A + C ) - D ) + B )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A + B ) + C ) =
     ( ( A + C ) + B ) by (rule MMI_add23t)
   from S1 have S2: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) + C ) = ( ( A + C ) + B ) by (rule MMI_3expa)
   from S2 have S3: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( A + B ) + C ) = ( ( A + C ) + B )
     by (rule MMI_adantrr)
   from S3 have S4: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( ( A + B ) + C ) - D ) =
     ( ( ( A + C ) + B ) - D ) by (rule MMI_opreq1d)
   have S5: ( ( A + C ) ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ ) ⟶
```

```
    ( ( ( A + C ) + B ) - D ) =
    ( ( ( A + C ) - D ) + B ) by (rule MMI_addsubt)
  from S5 have S6: ( ( A + C ) ∈ ℂ ∧ ( B ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
    ( ( ( A + C ) + B ) - D ) =
    ( ( ( A + C ) - D ) + B ) by (rule MMI_3expb)
  have S7: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + C ) ∈ ℂ by (rule MMI_axaddcl)
  from S6 S7 have S8: ( ( A ∈ ℂ ∧ C ∈ ℂ ) ∧ ( B ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

    ( ( ( A + C ) + B ) - D ) =
    ( ( ( A + C ) - D ) + B ) by (rule MMI_sylan)
  from S8 have S9: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

    ( ( ( A + C ) + B ) - D ) =
    ( ( ( A + C ) - D ) + B ) by (rule MMI_an4s)
  from S4 S9 show ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
    ( ( ( A + B ) + C ) - D ) =
    ( ( ( A + C ) - D ) + B ) by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_negneg: assumes A1: A ∈ ℂ
  shows ( - ( (-  A) ) ) = A
proof -
  from A1 have S1: A ∈ ℂ.
  from S1 have S2: ( (-  A) ) ∈ ℂ by (rule MMI_negcl)
  from S2 have S3: ( ( (-  A) ) + ( - ( (-  A) ) ) ) = 0
    by (rule MMI_negid)
  from S3 have S4: ( A + ( ( (-  A) ) + ( - ( (-  A) ) ) ) ) =
    ( A + 0 ) by (rule MMI_opreq2i)
  from A1 have S5: A ∈ ℂ.
  from S5 have S6: ( A + ( (-  A) ) ) = 0 by (rule MMI_negid)
  from S6 have S7: ( ( A + ( (-  A) ) ) + ( - ( (-  A) ) ) ) =
    ( 0 + ( - ( (-  A) ) ) ) by (rule MMI_opreq1i)
  from A1 have S8: A ∈ ℂ.
  from S2 have S9: ( (-  A) ) ∈ ℂ .
  from S2 have S10: ( (-  A) ) ∈ ℂ .
  from S10 have S11: ( - ( (-  A) ) ) ∈ ℂ by (rule MMI_negcl)
  from S8 S9 S11 have S12:
    ( ( A + ( (-  A) ) ) + ( - ( (-  A) ) ) ) =
    ( A + ( ( (-  A) ) + ( - ( (-  A) ) ) ) )
    by (rule MMI_addass)
  from S11 have S13: ( - ( (-  A) ) ) ∈ ℂ .
  from S13 have S14: ( 0 + ( - ( (-  A) ) ) ) =
    ( - ( (-  A) ) ) by (rule MMI_addid2)
  from S7 S12 S14 have S15:
    ( A + ( ( (-  A) ) + ( - ( (-  A) ) ) ) ) =
    ( - ( (-  A) ) ) by (rule MMI_3eqtr3)
  from A1 have S16: A ∈ ℂ.
  from S16 have S17: ( A + 0 ) = A by (rule MMI_addid1)
  from S4 S15 S17 show ( - ( (-  A) ) ) = A by (rule MMI_3eqtr3)
```

```
qed

lemma (in MMIsar0) MMI_subid: assumes A1: A ∈ ℂ
   shows ( A - A ) = 0
proof -
   from A1 have S1: A ∈ ℂ.
   from A1 have S2: A ∈ ℂ.
   from S1 S2 have S3: ( A + ( (-  A) ) ) = ( A - A )
     by (rule MMI_negsub)
   from A1 have S4: A ∈ ℂ.
   from S4 have S5: ( A + ( (-  A) ) ) = 0 by (rule MMI_negid)
   from S3 S5 show ( A - A ) = 0 by (rule MMI_eqtr3)
qed

lemma (in MMIsar0) MMI_subid1: assumes A1: A ∈ ℂ
   shows ( A - 0 ) = A
proof -
   from A1 have S1: A ∈ ℂ.
   from S1 have S2: ( 0 + A ) = A by (rule MMI_addid2)
   from A1 have S3: A ∈ ℂ.
   have S4: 0 ∈ ℂ by (rule MMI_0cn)
   from A1 have S5: A ∈ ℂ.
   from S3 S4 S5 have S6: ( A - 0 ) = A ←→ ( 0 + A ) = A
     by (rule MMI_subadd)
   from S2 S6 show ( A - 0 ) = A by (rule MMI_mpbir)
qed

lemma (in MMIsar0) MMI_negnegt:
   shows A ∈ ℂ −→ ( - ( (-  A) ) ) = A
proof -
   have S1: A = if ( A ∈ ℂ , A , 0 ) −→ ( (-  A) ) =
     ( - if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_negeq)
   from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) −→ ( - ( (-  A) ) ) =
     ( - ( - if ( A ∈ ℂ , A , 0 ) ) ) by (rule MMI_negeqd)
   have S3: A = if ( A ∈ ℂ , A , 0 ) −→ A = if ( A ∈ ℂ , A , 0 )
     by (rule MMI_id)
   from S2 S3 have S4: A = if ( A ∈ ℂ , A , 0 ) −→
     ( ( - ( (-  A) ) ) = A ←→
     ( - ( - if ( A ∈ ℂ , A , 0 ) ) ) = if ( A ∈ ℂ , A , 0 ) )
     by (rule MMI_eqeq12d)
   have S5: 0 ∈ ℂ by (rule MMI_0cn)
   from S5 have S6: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S6 have S7: ( - ( - if ( A ∈ ℂ , A , 0 ) ) ) =
     if ( A ∈ ℂ , A , 0 ) by (rule MMI_negneg)
   from S4 S7 show A ∈ ℂ −→ ( - ( (-  A) ) ) = A by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_subnegt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) −→ ( A - ( (-  B) ) ) = ( A + B )
```

```
proof -
   have S1: ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ) ⟶
     ( A + ( - ( (-  B) ) ) ) = ( A - ( (-  B) ) )
     by (rule MMI_negsubt)
   have S2: B ∈ ℂ ⟶ ( (-  B) ) ∈ ℂ by (rule MMI_negclt)
   from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( A + ( - ( (-  B) ) ) ) = ( A - ( (-  B) ) )
     by (rule MMI_sylan2)
   have S4: B ∈ ℂ ⟶ ( - ( (-  B) ) ) = B by (rule MMI_negnegt)
   from S4 have S5: B ∈ ℂ ⟶ ( A + ( - ( (-  B) ) ) ) =
     ( A + B ) by (rule MMI_opreq2d)
   from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( A + ( - ( (-  B) ) ) ) = ( A + B ) by (rule MMI_adantl)
   from S3 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - ( (-  B) ) ) =
     ( A + B ) by (rule MMI_eqtr3d)
qed

lemma (in MMIsar0) MMI_subidt:
   shows A ∈ ℂ ⟶ ( A - A ) = 0
proof -
   have S1: ( A = if ( A ∈ ℂ , A , 0 ) ∧ A = if ( A ∈ ℂ , A , 0 ) )
⟶
     ( A - A ) = ( if ( A ∈ ℂ , A , 0 ) - if ( A ∈ ℂ , A , 0 ) )
     by (rule MMI_opreq12)
   from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( A - A ) = ( if ( A ∈ ℂ , A , 0 ) - if ( A ∈ ℂ , A , 0 ) )
     by (rule MMI_anidms)
   from S2 have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( ( A - A ) = 0 ⟷
     ( if ( A ∈ ℂ , A , 0 ) - if ( A ∈ ℂ , A , 0 ) ) = 0 )
     by (rule MMI_eqeq1d)
   have S4: 0 ∈ ℂ by (rule MMI_0cn)
   from S4 have S5: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S5 have S6:
     ( if ( A ∈ ℂ , A , 0 ) - if ( A ∈ ℂ , A , 0 ) ) = 0
     by (rule MMI_subid)
   from S3 S6 show A ∈ ℂ ⟶ ( A - A ) = 0 by (rule MMI_dedth)
qed


lemma (in MMIsar0) MMI_subid1t:
   shows A ∈ ℂ ⟶ ( A - 0 ) = A
proof -
   have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A - 0 ) =
     ( if ( A ∈ ℂ , A , 0 ) - 0 ) by (rule MMI_opreq1)
   have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶
     A = if ( A ∈ ℂ , A , 0 ) by (rule MMI_id)
   from S1 S2 have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶
     ( ( A - 0 ) = A ⟷ ( if ( A ∈ ℂ , A , 0 ) - 0 ) =
```

```
    if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_eqeq12d)
  have S4: 0 ∈ ℂ by (rule MMI_0cn)
  from S4 have S5: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
  from S5 have S6: ( if ( A ∈ ℂ , A , 0 ) - 0 ) =
    if ( A ∈ ℂ , A , 0 ) by (rule MMI_subid1)
  from S3 S6 show A ∈ ℂ ⟶ ( A - 0 ) = A by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_pncant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) = A
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) =
    ( A + ( B - B ) ) by (rule MMI_addsubasst)
  from S1 have S2: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ B ∈ ℂ ) ) ⟶
    ( ( A + B ) - B ) = ( A + ( B - B ) ) by (rule MMI_3expb)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) =
    ( A + ( B - B ) ) by (rule MMI_anabsan2)
  have S4: B ∈ ℂ ⟶ ( B - B ) = 0 by (rule MMI_subidt)
  from S4 have S5: B ∈ ℂ ⟶ ( A + ( B - B ) ) = ( A + 0 )
    by (rule MMI_opreq2d)
  have S6: A ∈ ℂ ⟶ ( A + 0 ) = A by (rule MMI_ax0id)
  from S5 S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( B - B ) ) = A
    by (rule MMI_sylan9eqr)
  from S3 S7 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) = A
    by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_pncan2t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - A ) = B
proof -
  have S1: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( B + A ) = ( A + B )
    by (rule MMI_axaddcom)
  from S1 have S2: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( ( B + A ) - A ) =
    ( ( A + B ) - A ) by (rule MMI_opreq1d)
  have S3: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( ( B + A ) - A ) = B
    by (rule MMI_pncant)
  from S2 S3 have S4: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶
    ( ( A + B ) - A ) = B by (rule MMI_eqtr3d)
  from S4 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - A ) = B
    by (rule MMI_ancoms)
qed

lemma (in MMIsar0) MMI_npcant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) + B ) = A
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( A + B ) - B ) = ( ( A - B ) + B )
    by (rule MMI_addsubt)
  from S1 have S2: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ B ∈ ℂ ) ) ⟶
```

```
    ( ( A + B ) - B ) = ( ( A - B ) + B ) by (rule MMI_3expb)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( A + B ) - B ) = ( ( A - B ) + B )
    by (rule MMI_anabsan2)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A + B ) - B ) = A
    by (rule MMI_pncant)
  from S3 S4 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) + B ) = A
    by (rule MMI_eqtr3d)
qed

lemma (in MMIsar0) MMI_npncant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - B ) + ( B - C ) ) = ( A - C )
proof -
  have S1: ( ( A - B ) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) - C ) =
    ( ( A - B ) + ( B - C ) ) by (rule MMI_addsubasst)
  have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ by (rule MMI_subclt)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( A - B ) ∈ ℂ by (rule MMI_3adant3)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ B ∈ ℂ by (rule MMI_3simp2)
  have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ C ∈ ℂ by (rule MMI_3simp3)
  from S1 S3 S4 S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) - C ) =
    ( ( A - B ) + ( B - C ) ) by (rule MMI_syl3anc)
  have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) + B ) = A
    by (rule MMI_npcant)
  from S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) - C ) = ( A - C )
    by (rule MMI_opreq1d)
  from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) - C ) = ( A - C )
    by (rule MMI_3adant3)
  from S6 S9 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A - B ) + ( B - C ) ) = ( A - C )
    by (rule MMI_eqtr3d)
qed

lemma (in MMIsar0) MMI_nppcant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( ( A - B ) + C ) + B ) = ( A + C )
proof -
  have S1: ( ( A - B ) ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( ( A - B ) + C ) + B ) =
    ( ( ( A - B ) + B ) + C ) by (rule MMI_add23t)
  have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ by (rule MMI_subclt)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ
    by (rule MMI_3adant3)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ C ∈ ℂ by (rule MMI_3simp3)
```

```
  have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ B ∈ ℂ by (rule MMI_3simp2)
  from S1 S3 S4 S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + C ) + B ) =
    ( ( ( A - B ) + B ) + C ) by (rule MMI_syl3anc)
  have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) + B ) = A
    by (rule MMI_npcant)
  from S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) + C ) = ( A + C )
    by (rule MMI_opreq1d)
  from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + B ) + C ) = ( A + C )
    by (rule MMI_3adant3)
  from S6 S9 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A - B ) + C ) + B ) = ( A + C ) by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_subneg: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
  shows ( A - ( (-  B) ) ) = ( A + B )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - ( (-  B) ) ) = ( A + B )
    by (rule MMI_subnegt)
  from S1 S2 S3 show ( A - ( (-  B) ) ) = ( A + B )
    by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_subeq0: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
  shows ( A - B ) = 0 ⟷ A = B
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from S1 S2 have S3: ( A + ( (-  B) ) ) = ( A - B )
    by (rule MMI_negsub)
  from S3 have S4: ( A + ( (-  B) ) ) = 0 ⟷ ( A - B ) = 0
    by (rule MMI_eqeq1i)
  have S5: ( A + ( (-  B) ) ) = 0 ⟶
    ( ( A + ( (-  B) ) ) + B ) = ( 0 + B ) by (rule MMI_opreq1)
  from S4 S5 have S6: ( A - B ) = 0 ⟶
    ( ( A + ( (-  B) ) ) + B ) = ( 0 + B ) by (rule MMI_sylbir)
  from A1 have S7: A ∈ ℂ.
  from A2 have S8: B ∈ ℂ.
  from S8 have S9: ( (-  B) ) ∈ ℂ by (rule MMI_negcl)
  from A2 have S10: B ∈ ℂ.
  from S7 S9 S10 have S11: ( ( A + ( (-  B) ) ) + B ) =
    ( ( A + B ) + ( (-  B) ) ) by (rule MMI_add23)
  from A1 have S12: A ∈ ℂ.
```

```
from A2 have S13: B ∈ ℂ.
from S9 have S14: ( (-  B) ) ∈ ℂ .
from S12 S13 S14 have S15: ( ( A + B ) + ( (-  B) ) ) =
  ( A + ( B + ( (-  B) ) ) ) by (rule MMI_addass)
from A2 have S16: B ∈ ℂ.
from S16 have S17: ( B + ( (-  B) ) ) = 0 by (rule MMI_negid)
from S17 have S18: ( A + ( B + ( (-  B) ) ) ) = ( A + 0 )
  by (rule MMI_opreq2i)
from A1 have S19: A ∈ ℂ.
from S19 have S20: ( A + 0 ) = A by (rule MMI_addid1)
from S18 S20 have S21: ( A + ( B + ( (-  B) ) ) ) = A
  by (rule MMI_eqtr)
from S11 S15 S21 have S22: ( ( A + ( (-  B) ) ) + B ) = A
  by (rule MMI_3eqtr)
from A2 have S23: B ∈ ℂ.
from S23 have S24: ( 0 + B ) = B by (rule MMI_addid2)
from S6 S22 S24 have S25: ( A - B ) = 0 ⟶ A = B
  by (rule MMI_3eqtr3g)
have S26: A = B ⟶ ( A - B ) = ( B - B ) by (rule MMI_opreq1)
from A2 have S27: B ∈ ℂ.
from S27 have S28: ( B - B ) = 0 by (rule MMI_subid)
from S26 S28 have S29: A = B ⟶ ( A - B ) = 0 by (rule MMI_syl6eq)
from S25 S29 show ( A - B ) = 0 ⟷ A = B by (rule MMI_impbi)
qed

lemma (in MMIsar0) MMI_neg11: assumes A1: A ∈ ℂ and
   A2: B ∈ ℂ
  shows ( (-  A) ) = ( (-  B) ) ⟷ A = B
proof -
  have S1: ( (-  A) ) = ( 0 - A ) by (rule MMI_df_neg)
  have S2: ( (-  B) ) = ( 0 - B ) by (rule MMI_df_neg)
  from S1 S2 have S3: ( (-  A) ) = ( (-  B) ) ⟷ ( 0 - A ) =
    ( 0 - B ) by (rule MMI_eqeq12i)
  have S4: 0 ∈ ℂ by (rule MMI_0cn)
  from A1 have S5: A ∈ ℂ.
  have S6: 0 ∈ ℂ by (rule MMI_0cn)
  from A2 have S7: B ∈ ℂ.
  from S6 S7 have S8: ( 0 - B ) ∈ ℂ by (rule MMI_subcl)
  from S4 S5 S8 have S9: ( 0 - A ) = ( 0 - B ) ⟷
    ( A + ( 0 - B ) ) = 0 by (rule MMI_subadd)
  from S2 have S10: ( (-  B) ) = ( 0 - B ) .
  from S10 have S11: ( A + ( (-  B) ) ) = ( A + ( 0 - B ) )
    by (rule MMI_opreq2i)
  from A1 have S12: A ∈ ℂ.
  from A2 have S13: B ∈ ℂ.
  from S12 S13 have S14: ( A + ( (-  B) ) ) = ( A - B )
    by (rule MMI_negsub)
  from S11 S14 have S15: ( A + ( 0 - B ) ) = ( A - B )
    by (rule MMI_eqtr3)
```

```
  from S15 have S16: ( A + ( 0 - B ) ) = 0 ⟷ ( A - B ) = 0
    by (rule MMI_eqeq1i)
  from A1 have S17: A ∈ ℂ.
  from A2 have S18: B ∈ ℂ.
  from S17 S18 have S19: ( A - B ) = 0 ⟷ A = B by (rule MMI_subeq0)
  from S16 S19 have S20: ( A + ( 0 - B ) ) = 0 ⟷ A = B
    by (rule MMI_bitr)
  from S3 S9 S20 show ( (-  A) ) = ( (-  B) ) ⟷ A = B by (rule MMI_3bitr)
qed



lemma (in MMIsar0) MMI_negcon1: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
  shows ( (-  A) ) = B ⟷ ( (-  B) ) = A
proof -
  from A1 have S1: A ∈ ℂ.
  from S1 have S2: ( - ( (-  A) ) ) = A by (rule MMI_negneg)
  from S2 have S3: ( - ( (-  A) ) ) = ( (-  B) ) ⟷ A = ( (-  B) )

    by (rule MMI_eqeq1i)
  from A1 have S4: A ∈ ℂ.
  from S4 have S5: ( (-  A) ) ∈ ℂ by (rule MMI_negcl)
  from A2 have S6: B ∈ ℂ.
  from S5 S6 have S7: ( - ( (-  A) ) ) =
    ( (-  B) ) ⟷ ( (-  A) ) = B by (rule MMI_neg11)
  have S8: A = ( (-  B) ) ⟷ ( (-  B) ) = A by (rule MMI_eqcom)
  from S3 S7 S8 show ( (-  A) ) = B ⟷ ( (-  B) ) = A by (rule MMI_3bitr3)
qed

lemma (in MMIsar0) MMI_negcon2: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
  shows A = ( (-  B) ) ⟷ B = ( (-  A) )
proof -
  from A2 have S1: B ∈ ℂ.
  from A1 have S2: A ∈ ℂ.
  from S1 S2 have S3: ( (-  B) ) = A ⟷ ( (-  A) ) = B
    by (rule MMI_negcon1)
  have S4: A = ( (-  B) ) ⟷ ( (-  B) ) = A by (rule MMI_eqcom)
  have S5: B = ( (-  A) ) ⟷ ( (-  A) ) = B by (rule MMI_eqcom)
  from S3 S4 S5 show A = ( (-  B) ) ⟷ B = ( (-  A) ) by (rule MMI_3bitr4)
qed

lemma (in MMIsar0) MMI_neg11t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( (-  A) ) = ( (-  B) ) ⟷ A = B )
proof -
  have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( (-  A) ) =
    ( - if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_negeq)
  from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( ( (-  A) ) =
```

```
      ( (-  B) ) ⟷ ( - if ( A ∈ ℂ , A , 0 ) ) = ( (-  B) ) )
      by (rule MMI_eqeq1d)
   have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A = B ⟷
      if ( A ∈ ℂ , A , 0 ) = B ) by (rule MMI_eqeq1)
   from S2 S3 have S4: A = if ( A ∈ ℂ , A , 0 ) ⟶
      ( ( ( (-  A) ) = ( (-  B) ) ⟷ A = B ) ⟷
      ( ( - if ( A ∈ ℂ , A , 0 ) ) = ( (-  B) ) ⟷
      if ( A ∈ ℂ , A , 0 ) = B ) ) by (rule MMI_bibi12d)
   have S5: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( (-  B) ) =
      ( - if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_negeq)
   from S5 have S6: B = if ( B ∈ ℂ , B , 0 ) ⟶
      ( ( - if ( A ∈ ℂ , A , 0 ) ) = ( (-  B) ) ⟷
      ( - if ( A ∈ ℂ , A , 0 ) ) = ( - if ( B ∈ ℂ , B , 0 ) ) )
      by (rule MMI_eqeq2d)
   have S7: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( if ( A ∈ ℂ , A , 0 ) = B
⟷
      if ( A ∈ ℂ , A , 0 ) = if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_eqeq2)
   from S6 S7 have S8: B = if ( B ∈ ℂ , B , 0 ) ⟶
      ( ( ( - if ( A ∈ ℂ , A , 0 ) ) = ( (-  B) ) ⟷
      if ( A ∈ ℂ , A , 0 ) = B ) ⟷ ( ( - if ( A ∈ ℂ , A , 0 ) ) =
      ( - if ( B ∈ ℂ , B , 0 ) ) ⟷ if ( A ∈ ℂ , A , 0 ) =
      if ( B ∈ ℂ , B , 0 ) ) ) by (rule MMI_bibi12d)
   have S9: 0 ∈ ℂ by (rule MMI_0cn)
   from S9 have S10: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   have S11: 0 ∈ ℂ by (rule MMI_0cn)
   from S11 have S12: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S10 S12 have S13: ( - if ( A ∈ ℂ , A , 0 ) ) =
      ( - if ( B ∈ ℂ , B , 0 ) ) ⟷ if ( A ∈ ℂ , A , 0 ) =
      if ( B ∈ ℂ , B , 0 ) by (rule MMI_neg11)
   from S4 S8 S13 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( (-  A) ) =
      ( (-  B) ) ⟷ A = B ) by (rule MMI_dedth2h)
qed

lemma (in MMIsar0) MMI_negcon1t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( (-  A) ) = B ⟷ ( (-  B) ) = A )
proof -
   have S1: ( ( (-  A) ) ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( - ( (-  A) ) ) =
      ( (-  B) ) ⟷ ( (-  A) ) = B ) by (rule MMI_neg11t)
   have S2: A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI_negclt)
   from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( - ( (-  A) ) ) =
      ( (-  B) ) ⟷ ( (-  A) ) = B ) by (rule MMI_sylan)
   have S4: A ∈ ℂ ⟶ ( - ( (-  A) ) ) = A by (rule MMI_negnegt)
   from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( - ( (-  A) ) ) = A
      by (rule MMI_adantr)
   from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( - ( (-  A) ) ) =
      ( (-  B) ) ⟷ A = ( (-  B) ) ) by (rule MMI_eqeq1d)
   from S3 S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( (-  A) ) = B ⟷ A
=
      ( (-  B) ) ) by (rule MMI_bitr3d)
```

```
   have S8: A = ( (-  B) ) ⟷ ( (-  B) ) = A by (rule MMI_eqcom)
   from S7 S8 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( (-  A) ) = B ⟷
     ( (-  B) ) = A ) by (rule MMI_syl6bb)
qed

lemma (in MMIsar0) MMI_negcon2t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A = ( (-  B) ) ⟷ B = ( (-  A) ) )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( (-  A) ) = B ⟷ ( (-  B) ) =
A )
     by (rule MMI_negcon1t)
   have S2: A = ( (-  B) ) ⟷ ( (-  B) ) = A by (rule MMI_eqcom)
   from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A = ( (-  B) ) ⟷
     ( (-  A) ) = B ) by (rule MMI_syl6rbbrA)
   have S4: ( (-  A) ) = B ⟷ B = ( (-  A) ) by (rule MMI_eqcom)
   from S3 S4 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A = ( (-  B) ) ⟷ B =
     ( (-  A) ) ) by (rule MMI_syl6bb)
qed

lemma (in MMIsar0) MMI_subcant:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A - B ) =
  ( A - C ) ⟷ B = C )
proof -
   have S1: ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ∧ ( - C ) ∈ ℂ ) ⟶
     ( ( A + ( (-  B) ) ) = ( A + ( - C ) ) ⟷
     ( (-  B) ) = ( - C ) ) by (rule MMI_addcant)
   have S2: C ∈ ℂ ⟶ ( - C ) ∈ ℂ by (rule MMI_negclt)
   from S1 S2 have S3: ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + ( (-  B) ) ) = ( A + ( - C ) ) ⟷
     ( (-  B) ) = ( - C ) ) by (rule MMI_syl3an3)
   have S4: B ∈ ℂ ⟶ ( (-  B) ) ∈ ℂ by (rule MMI_negclt)
   from S3 S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + ( (-  B) ) ) = ( A + ( - C ) ) ⟷
     ( (-  B) ) = ( - C ) ) by (rule MMI_syl3an2)
   have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( (-  B) ) ) = ( A - B )
     by (rule MMI_negsubt)
   from S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( A + ( (-  B) ) ) = ( A - B ) by (rule MMI_3adant3)
   have S8: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + ( - C ) ) = ( A - C )
     by (rule MMI_negsubt)
   from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( A + ( - C ) ) = ( A - C ) by (rule MMI_3adant2)
   from S7 S9 have S10: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + ( (-  B) ) ) = ( A + ( - C ) ) ⟷
     ( A - B ) = ( A - C ) ) by (rule MMI_eqeq12d)
   have S11: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( (-  B) ) = ( - C ) ⟷ B = C
)
     by (rule MMI_neg11t)
   from S11 have S12: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
```

```
    ( ( (-  B) ) = ( - C ) ⟷ B = C ) by (rule MMI_3adant1)
  from S5 S10 S12 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A - B ) = ( A - C ) ⟷ B = C ) by (rule MMI_3bitr3d)
qed

lemma (in MMIsar0) MMI_subcan2t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - C ) = ( B - C ) ⟷ A = B )
proof -
  have S1: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A + ( - C ) ) = ( A - C )
    by (rule MMI_negsubt)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( A + ( - C ) ) = ( A - C ) by (rule MMI_3adant2)
  have S3: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B + ( - C ) ) = ( B - C )
    by (rule MMI_negsubt)
  from S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( B + ( - C ) ) = ( B - C ) by (rule MMI_3adant1)
  from S2 S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + ( - C ) ) = ( B + ( - C ) ) ⟷ ( A - C ) =
    ( B - C ) ) by (rule MMI_eqeq12d)
  have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( - C ) ∈ ℂ ) ⟶
    ( ( A + ( - C ) ) = ( B + ( - C ) ) ⟷ A = B )
    by (rule MMI_addcan2t)
  have S7: C ∈ ℂ ⟶ ( - C ) ∈ ℂ by (rule MMI_negclt)
  from S6 S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A + ( - C ) ) = ( B + ( - C ) ) ⟷ A = B )
    by (rule MMI_syl3an3)
  from S5 S8 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A - C ) = ( B - C ) ⟷ A = B ) by (rule MMI_bitr3d)
qed

lemma (in MMIsar0) MMI_subcan: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
  shows ( A - B ) = ( A - C ) ⟷ B = C
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from A3 have S3: C ∈ ℂ.
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A - B ) = ( A - C ) ⟷
B = C ) by (rule MMI_subcant)
  from S1 S2 S3 S4 show ( A - B ) = ( A - C ) ⟷ B = C
    by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_subcan2: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
  shows ( A - C ) = ( B - C ) ⟷ A = B
```

```
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from A3 have S3: C ∈ ℂ.
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A - C ) = ( B - C ) ⟷ A = B ) by (rule MMI_subcan2t)
  from S1 S2 S3 S4 show ( A - C ) = ( B - C ) ⟷ A = B
    by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_subeq0t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A - B ) = 0 ⟷ A = B )
proof -
  have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A - B ) =
    ( if ( A ∈ ℂ , A , 0 ) - B ) by (rule MMI_opreq1)
  from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( ( A - B ) = 0 ⟷

    ( if ( A ∈ ℂ , A , 0 ) - B ) = 0 ) by (rule MMI_eqeq1d)
  have S3: A = if ( A ∈ ℂ , A , 0 ) ⟶ ( A = B ⟷
    if ( A ∈ ℂ , A , 0 ) = B ) by (rule MMI_eqeq1)
  from S2 S3 have S4: A = if ( A ∈ ℂ , A , 0 ) ⟶
    ( ( ( A - B ) = 0 ⟷ A = B ) ⟷
    ( ( if ( A ∈ ℂ , A , 0 ) - B ) = 0 ⟷
    if ( A ∈ ℂ , A , 0 ) = B ) ) by (rule MMI_bibi12d)
  have S5: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( if ( A ∈ ℂ , A , 0 ) - B ) =
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) )
    by (rule MMI_opreq2)
  from S5 have S6: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( ( if ( A ∈ ℂ , A , 0 ) - B ) = 0 ⟷
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = 0 )
    by (rule MMI_eqeq1d)
  have S7: B = if ( B ∈ ℂ , B , 0 ) ⟶ ( if ( A ∈ ℂ , A , 0 ) = B
⟷
    if ( A ∈ ℂ , A , 0 ) = if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_eqeq2)
  from S6 S7 have S8: B = if ( B ∈ ℂ , B , 0 ) ⟶
    ( ( ( if ( A ∈ ℂ , A , 0 ) - B ) = 0 ⟷
    if ( A ∈ ℂ , A , 0 ) = B ) ⟷
    ( ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = 0 ⟷
    if ( A ∈ ℂ , A , 0 ) = if ( B ∈ ℂ , B , 0 ) ) )
    by (rule MMI_bibi12d)
  have S9: 0 ∈ ℂ by (rule MMI_0cn)
  from S9 have S10: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
  have S11: 0 ∈ ℂ by (rule MMI_0cn)
  from S11 have S12: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
  from S10 S12 have S13:
    ( if ( A ∈ ℂ , A , 0 ) - if ( B ∈ ℂ , B , 0 ) ) = 0 ⟷
    if ( A ∈ ℂ , A , 0 ) = if ( B ∈ ℂ , B , 0 )
    by (rule MMI_subeq0)
```

```
   from S4 S8 S13 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( ( A - B ) = 0 ⟷ A = B ) by (rule MMI_dedth2h)
qed



lemma (in MMIsar0) MMI_neg0:
   shows ( - 0 ) = 0
proof -
   have S1: ( - 0 ) = ( 0 - 0 ) by (rule MMI_df_neg)
   have S2: 0 ∈ ℂ by (rule MMI_0cn)
   from S2 have S3: ( 0 - 0 ) = 0 by (rule MMI_subid)
   from S1 S3 show ( - 0 ) = 0 by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_renegcl: assumes A1: A ∈ ℝ
   shows ( (-  A) ) ∈ ℝ
proof -
   from A1 have S1: A ∈ ℝ.
   have S2: A ∈ ℝ ⟶ ( ∃ x ∈ ℝ . ( A + x ) = 0 ) by (rule MMI_axrnegex)
   from S1 S2 have S3: ∃ x ∈ ℝ . ( A + x ) = 0 by (rule MMI_ax_mp)
   have S4: ( ∃ x ∈ ℝ . ( A + x ) = 0 ) ⟷
     ( ∃ x . ( x ∈ ℝ ∧ ( A + x ) = 0 ) ) by (rule MMI_df_rex)
   from S3 S4 have S5: ∃ x . ( x ∈ ℝ ∧ ( A + x ) = 0 )
     by (rule MMI_mpbi)
   { fix x
     have S6: x ∈ ℝ ⟶ x ∈ ℂ by (rule MMI_recnt)
     have S7: 0 ∈ ℂ by (rule MMI_0cn)
     from A1 have S8: A ∈ ℝ.
     from S8 have S9: A ∈ ℂ by (rule MMI_recn)
     have S10: ( 0 ∈ ℂ ∧ A ∈ ℂ ∧ x ∈ ℂ ) ⟶ ( ( 0 - A ) = x ⟷
       ( A + x ) = 0 ) by (rule MMI_subaddt)
     from S7 S9 S10 have S11: x ∈ ℂ ⟶ ( ( 0 - A ) = x ⟷
       ( A + x ) = 0 ) by (rule MMI_mp3an12)
     from S6 S11 have S12: x ∈ ℝ ⟶ ( ( 0 - A ) = x ⟷
       ( A + x ) = 0 ) by (rule MMI_syl)
     have S13: ( (-  A) ) = ( 0 - A ) by (rule MMI_df_neg)
     from S13 have S14: ( (-  A) ) = x ⟷ ( 0 - A ) = x
       by (rule MMI_eqeq1i)
     from S12 S14 have S15: x ∈ ℝ ⟶ ( ( (-  A) ) = x ⟷
       ( A + x ) = 0 ) by (rule MMI_syl5bb)
     have S16: x ∈ ℝ ⟶ ( ( (-  A) ) = x ⟶ ( (-  A) ) ∈ ℝ )
       by (rule MMI_eleq1a)
     from S15 S16 have S17: x ∈ ℝ ⟶ ( ( A + x ) = 0 ⟶
       ( (-  A) ) ∈ ℝ ) by (rule MMI_sylbird)
     from S17 have ( x ∈ ℝ ∧ ( A + x ) = 0 ) ⟶ ( (-  A) ) ∈ ℝ
       by (rule MMI_imp)
     } then have S18:
  ∀x . ( x ∈ ℝ ∧ ( A + x ) = 0 ) ⟶ ( (-  A) ) ∈ ℝ
```

```
      by auto
   from S18 have S19: ( ∃ x . ( x ∈ ℝ ∧ ( A + x ) = 0 ) ) ⟶
     ( (-  A) ) ∈ ℝ by (rule MMI_19_23aiv)
   from S5 S19 show ( (-  A) ) ∈ ℝ by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_renegclt:
   shows A ∈ ℝ ⟶ ( (-  A) ) ∈ ℝ
proof -
   have S1: A = if ( A ∈ ℝ , A , 1 ) ⟶ ( (-  A) ) =
     ( - if ( A ∈ ℝ , A , 1 ) ) by (rule MMI_negeq)
   from S1 have S2: A = if ( A ∈ ℝ , A , 1 ) ⟶ ( ( (-  A) ) ∈ ℝ ⟷

     ( - if ( A ∈ ℝ , A , 1 ) ) ∈ ℝ ) by (rule MMI_eleq1d)
   have S3: 1 ∈ ℝ by (rule MMI_ax1re)
   from S3 have S4: if ( A ∈ ℝ , A , 1 ) ∈ ℝ by (rule MMI_elimel)
   from S4 have S5: ( - if ( A ∈ ℝ , A , 1 ) ) ∈ ℝ by (rule MMI_renegcl)
   from S2 S5 show A ∈ ℝ ⟶ ( (-  A) ) ∈ ℝ by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_resubclt:
   shows ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A - B ) ∈ ℝ
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + ( (-  B) ) ) = ( A - B )
     by (rule MMI_negsubt)
   have S2: A ∈ ℝ ⟶ A ∈ ℂ by (rule MMI_recnt)
   have S3: B ∈ ℝ ⟶ B ∈ ℂ by (rule MMI_recnt)
   from S1 S2 S3 have S4: ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A + ( (-  B) ) )
=
     ( A - B ) by (rule MMI_syl2an)
   have S5: ( A ∈ ℝ ∧ ( (-  B) ) ∈ ℝ ) ⟶ ( A + ( (-  B) ) ) ∈ ℝ
     by (rule MMI_axaddrcl)
   have S6: B ∈ ℝ ⟶ ( (-  B) ) ∈ ℝ by (rule MMI_renegclt)
   from S5 S6 have S7: ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A + ( (-  B) ) ) ∈ ℝ

     by (rule MMI_sylan2)
   from S4 S7 show ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A - B ) ∈ ℝ
     by (rule MMI_eqeltrrd)
qed

lemma (in MMIsar0) MMI_resubcl: assumes A1: A ∈ ℝ and
    A2: B ∈ ℝ
   shows ( A - B ) ∈ ℝ
proof -
   from A1 have S1: A ∈ ℝ.
   from A2 have S2: B ∈ ℝ.
   have S3: ( A ∈ ℝ ∧ B ∈ ℝ ) ⟶ ( A - B ) ∈ ℝ by (rule MMI_resubclt)
   from S1 S2 S3 show ( A - B ) ∈ ℝ by (rule MMI_mp2an)
qed
```

```
lemma (in MMIsar0) MMI_0re:
   shows 0 ∈ ℝ
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   from S1 have S2: ( 1 - 1 ) = 0 by (rule MMI_subid)
   have S3: 1 ∈ ℝ by (rule MMI_ax1re)
   have S4: 1 ∈ ℝ by (rule MMI_ax1re)
   from S3 S4 have S5: ( 1 - 1 ) ∈ ℝ by (rule MMI_resubcl)
   from S2 S5 show 0 ∈ ℝ by (rule MMI_eqeltrr)
qed
```

```
lemma (in MMIsar0) MMI_mulid2t:
   shows A ∈ ℂ ⟶ ( 1 · A ) = A
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   have S2: ( 1 ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( 1 · A ) = ( A · 1 )
     by (rule MMI_axmulcom)
   from S1 S2 have S3: A ∈ ℂ ⟶ ( 1 · A ) = ( A · 1 ) by (rule MMI_mpan)
   have S4: A ∈ ℂ ⟶ ( A · 1 ) = A by (rule MMI_ax1id)
   from S3 S4 show A ∈ ℂ ⟶ ( 1 · A ) = A by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_mul12t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · ( B · C ) ) =
  ( B · ( A · C ) )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) = ( B · A )
     by (rule MMI_axmulcom)
   from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( ( A · B ) · C ) = ( ( B · A ) · C ) by (rule MMI_opreq1d)
   from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A · B ) · C ) = ( ( B · A ) · C ) by (rule MMI_3adant3)
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A · B ) · C ) = ( A · ( B · C ) ) by (rule MMI_axmulass)
   have S5: ( B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( B · A ) · C ) = ( B · ( A · C ) ) by (rule MMI_axmulass)
   from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( B · A ) · C ) = ( B · ( A · C ) ) by (rule MMI_3com12)
   from S3 S4 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( A · ( B · C ) ) = ( B · ( A · C ) ) by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_mul23t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A · B ) · C ) =
  ( ( A · C ) · B )
proof -
```

```
  have S1: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B · C ) = ( C · B )
    by (rule MMI_axmulcom)
  from S1 have S2: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · ( B · C ) ) =
    ( A · ( C · B ) ) by (rule MMI_opreq2d)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · ( B · C ) )
=
    ( A · ( C · B ) ) by (rule MMI_3adant1)
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A · B ) · C ) =
    ( A · ( B · C ) ) by (rule MMI_axmulass)
  have S5: ( A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( A · C ) · B ) =
    ( A · ( C · B ) ) by (rule MMI_axmulass)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A · C ) · B ) = ( A · ( C · B ) ) by (rule MMI_3com23)
  from S3 S4 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A · B ) · C ) = ( ( A · C ) · B ) by (rule MMI_3eqtr4d)
qed

lemma (in MMIsar0) MMI_mul4t:
  shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
 ( ( A · B ) · ( C · D ) ) = ( ( A · C ) · ( B · D ) )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( A · B ) · C ) = ( ( A · C ) · B ) by (rule MMI_mul23t)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
    ( ( ( A · B ) · C ) · D ) = ( ( ( A · C ) · B ) · D )
    by (rule MMI_opreq1d)
  from S2 have S3: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
    ( ( ( A · B ) · C ) · D ) = ( ( ( A · C ) · B ) · D )
    by (rule MMI_3expa)
  from S3 have S4: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

    ( ( ( A · B ) · C ) · D ) = ( ( ( A · C ) · B ) · D )
    by (rule MMI_adantrr)
  have S5: ( ( A · B ) ∈ ℂ ∧ C ∈ ℂ ∧ D ∈ ℂ ) ⟶
    ( ( ( A · B ) · C ) · D ) = ( ( A · B ) · ( C · D ) )
    by (rule MMI_axmulass)
  from S5 have S6: ( ( A · B ) ∈ ℂ ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
    ( ( ( A · B ) · C ) · D ) = ( ( A · B ) · ( C · D ) ) by (rule MMI_3expb)
  have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) ∈ ℂ by (rule MMI_axmulcl)
  from S6 S7 have S8: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

    ( ( ( A · B ) · C ) · D ) = ( ( A · B ) · ( C · D ) ) by (rule MMI_sylan)
  have S9: ( ( A · C ) ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ ) ⟶
    ( ( ( A · C ) · B ) · D ) = ( ( A · C ) · ( B · D ) )
    by (rule MMI_axmulass)
  from S9 have S10: ( ( A · C ) ∈ ℂ ∧ ( B ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
    ( ( ( A · C ) · B ) · D ) = ( ( A · C ) · ( B · D ) )
    by (rule MMI_3expb)
  have S11: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · C ) ∈ ℂ by (rule MMI_axmulcl)
```

```
   from S10 S11 have S12: ( ( A ∈ ℂ ∧ C ∈ ℂ ) ∧ ( B ∈ ℂ ∧ D ∈ ℂ )
) ⟶
     ( ( ( A · C ) · B ) · D ) = ( ( A · C ) · ( B · D ) )
     by (rule MMI_sylan)
   from S12 have S13: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( ( A · C ) · B ) · D ) = ( ( A · C ) · ( B · D ) )
     by (rule MMI_an4s)
   from S4 S8 S13 show ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( A · B ) · ( C · D ) ) = ( ( A · C ) · ( B · D ) )
     by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_muladdt:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
  ( ( A + B ) · ( C + D ) ) =
  ( ( ( A · C ) + ( D · B ) ) + ( ( A · D ) + ( C · B ) ) )
proof -
   have S1: ( ( A + B ) ∈ ℂ ∧ C ∈ ℂ ∧ D ∈ ℂ ) ⟶
     ( ( A + B ) · ( C + D ) ) =
     ( ( ( A + B ) · C ) + ( ( A + B ) · D ) )
     by (rule MMI_axdistr)
   have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) ∈ ℂ by (rule MMI_axaddcl)
   from S2 have S3: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( A + B ) ∈ ℂ by (rule MMI_adantr)
   have S4: ( C ∈ ℂ ∧ D ∈ ℂ ) ⟶ C ∈ ℂ by (rule MMI_pm3_26)
   from S4 have S5: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
C ∈ ℂ
     by (rule MMI_adantl)
   have S6: ( C ∈ ℂ ∧ D ∈ ℂ ) ⟶ D ∈ ℂ by (rule MMI_pm3_27)
   from S6 have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
D ∈ ℂ
     by (rule MMI_adantl)
   from S1 S3 S5 S7 have S8:
     ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( A + B ) · ( C + D ) ) =
     ( ( ( A + B ) · C ) + ( ( A + B ) · D ) )
     by (rule MMI_syl3anc)
   have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) · C ) = ( ( A · C ) + ( B · C ) )
     by (rule MMI_adddirt)
   from S9 have S10: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
     ( ( A + B ) · C ) = ( ( A · C ) + ( B · C ) )
     by (rule MMI_3expa)
   from S10 have S11: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

     ( ( A + B ) · C ) = ( ( A · C ) + ( B · C ) )
```

by (rule MMI_adantrr)
have S12: (A ∈ ℂ ∧ B ∈ ℂ ∧ D ∈ ℂ) ⟶
((A + B) · D) = ((A · D) + (B · D))
by (rule MMI_adddirt)
from S12 have S13: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ D ∈ ℂ) ⟶
((A + B) · D) = ((A · D) + (B · D))
by (rule MMI_3expa)
from S13 have S14: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A + B) · D) = ((A · D) + (B · D))
by (rule MMI_adantrl)
from S11 S14 have S15: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)
) ⟶
(((A + B) · C) + ((A + B) · D)) =
(((A · C) + (B · C)) + ((A · D) + (B · D)))
by (rule MMI_opreq12d)
have S16:
((A · C) ∈ ℂ ∧ (B · C) ∈ ℂ ∧
((A · D) + (B · D)) ∈ ℂ) ⟶
(((A · C) + (B · C)) + ((A · D) + (B · D))) =
(((A · C) + ((A · D) + (B · D))) + (B · C))
by (rule MMI_add23t)
have S17: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ by (rule MMI_axmulcl)
from S17 have S18: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(A · C) ∈ ℂ by (rule MMI_ad2ant2r)
have S19: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) ∈ ℂ by (rule MMI_axmulcl)
from S19 have S20: (B ∈ ℂ ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(B · C) ∈ ℂ by (rule MMI_adantrr)
from S20 have S21: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(B · C) ∈ ℂ by (rule MMI_adantll)
have S22: ((A · D) ∈ ℂ ∧ (B · D) ∈ ℂ) ⟶
((A · D) + (B · D)) ∈ ℂ by (rule MMI_axaddcl)
have S23: (A ∈ ℂ ∧ D ∈ ℂ) ⟶ (A · D) ∈ ℂ by (rule MMI_axmulcl)
have S24: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (B · D) ∈ ℂ by (rule MMI_axmulcl)
from S22 S23 S24 have S25:
((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · D) + (B · D)) ∈ ℂ by (rule MMI_syl2an)
from S25 have S26: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ D ∈ ℂ) ⟶
((A · D) + (B · D)) ∈ ℂ by (rule MMI_anandirs)
from S26 have S27: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A · D) + (B · D)) ∈ ℂ by (rule MMI_adantrl)
from S16 S18 S21 S27 have S28:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) + (B · C)) + ((A · D) + (B · D))) =
(((A · C) + ((A · D) + (B · D))) + (B · C))
by (rule MMI_syl3anc)

have S29: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (B · D) = (D · B)
  by (rule MMI_axmulcom)
from S29 **have** S30: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

  (B · D) = (D · B) **by** (rule MMI_ad2ant2l)
from S30 **have** S31: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

  (((A · C) + (A · D)) + (B · D)) =
  (((A · C) + (A · D)) + (D · B))
  by (rule MMI_opreq2d)
have S32: ((A · C) ∈ ℂ ∧ (A · D) ∈ ℂ ∧ (B · D) ∈ ℂ) ⟶
  (((A · C) + (A · D)) + (B · D)) =
  ((A · C) + ((A · D) + (B · D)))
  by (rule MMI_axaddass)
from S18 **have** S33:
  ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶ (A · C) ∈ ℂ .
from S23 **have** S34: (A ∈ ℂ ∧ D ∈ ℂ) ⟶ (A · D) ∈ ℂ .
from S34 **have** S35: (A ∈ ℂ ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
  (A · D) ∈ ℂ **by** (rule MMI_adantrl)
from S35 **have** S36: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

  (A · D) ∈ ℂ **by** (rule MMI_adantlr)
from S24 **have** S37: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (B · D) ∈ ℂ .
from S37 **have** S38: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

  (B · D) ∈ ℂ **by** (rule MMI_ad2ant2l)
from S32 S33 S36 S38 **have** S39:
  ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
  (((A · C) + (A · D)) + (B · D)) =
  ((A · C) + ((A · D) + (B · D))) **by** (rule MMI_syl3anc)
have S40: ((A · C) ∈ ℂ ∧ (A · D) ∈ ℂ ∧ (D · B) ∈ ℂ) ⟶
  (((A · C) + (A · D)) + (D · B)) =
  (((A · C) + (D · B)) + (A · D)) **by** (rule MMI_add23t)
from S18 **have** S41:
  ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶ (A · C) ∈ ℂ .
from S36 **have** S42: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

  (A · D) ∈ ℂ .
have S43: (D ∈ ℂ ∧ B ∈ ℂ) ⟶ (D · B) ∈ ℂ **by** (rule MMI_axmulcl)
from S43 **have** S44: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (D · B) ∈ ℂ
  by (rule MMI_ancoms)
from S44 **have** S45: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

  (D · B) ∈ ℂ **by** (rule MMI_ad2ant2l)
from S40 S41 S42 S45 **have** S46:
  ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
  (((A · C) + (A · D)) + (D · B)) =
  (((A · C) + (D · B)) + (A · D)) **by** (rule MMI_syl3anc)
from S31 S39 S46 **have** S47:

((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · C) + ((A · D) + (B · D))) =
(((A · C) + (D · B)) + (A · D)) **by** (rule MMI_3eqtr3d)
have S48: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B)
by (rule MMI_axmulcom)
from S48 **have** S49: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶

(B · C) = (C · B) **by** (rule MMI_adantl)
from S49 **have** S50: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(B · C) = (C · B) **by** (rule MMI_an42s)
from S47 S50 **have** S51: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)
) ⟶
(((A · C) + ((A · D) + (B · D))) + (B · C)) =
((((A · C) + (D · B)) + (A · D)) + (C · B))
by (rule MMI_opreq12d)
have S52:
(((A · C) + (D · B)) ∈ ℂ ∧ (A · D) ∈ ℂ ∧
(C · B) ∈ ℂ) ⟶
((((A · C) + (D · B)) + (A · D)) + (C · B)) =
(((A · C) + (D · B)) + ((A · D) + (C · B)))
by (rule MMI_axaddass)
have S53: ((A · C) ∈ ℂ ∧ (D · B) ∈ ℂ) ⟶
((A · C) + (D · B)) ∈ ℂ **by** (rule MMI_axaddcl)
from S17 **have** S54: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ .
from S44 **have** S55: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (D · B) ∈ ℂ .
from S53 S54 S55 **have** S56:
((A ∈ ℂ ∧ C ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · C) + (D · B)) ∈ ℂ **by** (rule MMI_syl2an)
from S56 **have** S57: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A · C) + (D · B)) ∈ ℂ **by** (rule MMI_an4s)
from S36 **have** S58: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(A · D) ∈ ℂ .
have S59: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C · B) ∈ ℂ **by** (rule MMI_axmulcl)
from S59 **have** S60: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (C · B) ∈ ℂ
by (rule MMI_ancoms)
from S60 **have** S61: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶

(C · B) ∈ ℂ **by** (rule MMI_adantl)
from S61 **have** S62: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(C · B) ∈ ℂ **by** (rule MMI_an42s)
from S52 S57 S58 S62 **have** S63:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((((A · C) + (D · B)) + (A · D)) + (C · B)) =
(((A · C) + (D · B)) + ((A · D) + (C · B)))
by (rule MMI_syl3anc)

```
   from S28 S51 S63 have S64:
     ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( ( A · C ) + ( B · C ) ) + ( ( A · D ) + ( B · D ) ) ) =
     ( ( ( A · C ) + ( D · B ) ) + ( ( A · D ) + ( C · B ) ) )
     by (rule MMI_3eqtrd)
   from S8 S15 S64 show ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) )
⟶
     ( ( A + B ) · ( C + D ) ) =
     ( ( ( A · C ) + ( D · B ) ) + ( ( A · D ) + ( C · B ) ) )
     by (rule MMI_3eqtrd)
qed
```

```
lemma (in MMIsar0) MMI_muladd11t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( 1 + A ) · ( 1 + B ) ) =
  ( ( 1 + A ) + ( B + ( A · B ) ) )
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   have S2: ( ( 1 + A ) ∈ ℂ ∧ 1 ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( ( 1 + A ) · ( 1 + B ) ) =
     ( ( ( 1 + A ) · 1 ) + ( ( 1 + A ) · B ) )
     by (rule MMI_axdistr)
   from S1 S2 have S3: ( ( 1 + A ) ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( ( 1 + A ) · ( 1 + B ) ) =
     ( ( ( 1 + A ) · 1 ) + ( ( 1 + A ) · B ) )
     by (rule MMI_mp3an2)
   have S4: 1 ∈ ℂ by (rule MMI_1cn)
   have S5: ( 1 ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( 1 + A ) ∈ ℂ by (rule MMI_axaddcl)
   from S4 S5 have S6: A ∈ ℂ ⟶ ( 1 + A ) ∈ ℂ by (rule MMI_mpan)
   from S3 S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( ( 1 + A ) · ( 1 + B ) ) =
     ( ( ( 1 + A ) · 1 ) + ( ( 1 + A ) · B ) ) by (rule MMI_sylan)
   from S6 have S8: A ∈ ℂ ⟶ ( 1 + A ) ∈ ℂ .
   have S9: ( 1 + A ) ∈ ℂ ⟶ ( ( 1 + A ) · 1 ) = ( 1 + A )
     by (rule MMI_ax1id)
   from S8 S9 have S10: A ∈ ℂ ⟶ ( ( 1 + A ) · 1 ) = ( 1 + A )
     by (rule MMI_syl)
   from S10 have S11: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
     ( ( 1 + A ) · 1 ) = ( 1 + A ) by (rule MMI_adantr)
   have S12: 1 ∈ ℂ by (rule MMI_1cn)
   have S13: ( 1 ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( 1 + A ) · B ) =
     ( ( 1 · B ) + ( A · B ) ) by (rule MMI_adddirt)
   from S12 S13 have S14: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( 1 + A ) · B ) =
( ( 1 · B ) + ( A · B ) ) by (rule MMI_mp3an1)
   have S15: B ∈ ℂ ⟶ ( 1 · B ) = B by (rule MMI_mulid2t)
   from S15 have S16: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( 1 · B ) = B
     by (rule MMI_adantl)
   from S16 have S17:
```

```
    ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( 1 · B ) + ( A · B ) ) =
    ( B + ( A · B ) ) by (rule MMI_opreq1d)
  from S14 S17 have S18:
    ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( ( 1 + A ) · B ) =
    ( B + ( A · B ) ) by (rule MMI_eqtrd)
  from S11 S18 have S19: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( ( 1 + A ) · 1 ) + ( ( 1 + A ) · B ) ) =
    ( ( 1 + A ) + ( B + ( A · B ) ) ) by (rule MMI_opreq12d)
  from S7 S19 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
    ( ( 1 + A ) · ( 1 + B ) ) =
    ( ( 1 + A ) + ( B + ( A · B ) ) )
    by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_mul12: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
  shows ( A · ( B · C ) ) = ( B · ( A · C ) )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from S1 S2 have S3: ( A · B ) = ( B · A ) by (rule MMI_mulcom)
  from S3 have S4: ( ( A · B ) · C ) = ( ( B · A ) · C )
    by (rule MMI_opreq1i)
  from A1 have S5: A ∈ ℂ.
  from A2 have S6: B ∈ ℂ.
  from A3 have S7: C ∈ ℂ.
  from S5 S6 S7 have S8: ( ( A · B ) · C ) = ( A · ( B · C ) )
    by (rule MMI_mulass)
  from A2 have S9: B ∈ ℂ.
  from A1 have S10: A ∈ ℂ.
  from A3 have S11: C ∈ ℂ.
  from S9 S10 S11 have S12: ( ( B · A ) · C ) = ( B · ( A · C ) )
    by (rule MMI_mulass)
  from S4 S8 S12 show ( A · ( B · C ) ) = ( B · ( A · C ) )
    by (rule MMI_3eqtr3)
qed

lemma (in MMIsar0) MMI_mul23: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
  shows ( ( A · B ) · C ) = ( ( A · C ) · B )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from A3 have S3: C ∈ ℂ.
  have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( ( A · B ) · C ) =
    ( ( A · C ) · B ) by (rule MMI_mul23t)
  from S1 S2 S3 S4 show ( ( A · B ) · C ) = ( ( A · C ) · B )
```

```
     by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_mul4: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ and
    A4: D ∈ ℂ
   shows ( ( A · B ) · ( C · D ) ) = ( ( A · C ) · ( B · D ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI_pm3_2i)
   from A3 have S4: C ∈ ℂ.
   from A4 have S5: D ∈ ℂ.
   from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI_pm3_2i)
   have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( A · B ) · ( C · D ) ) = ( ( A · C ) · ( B · D ) )
     by (rule MMI_mul4t)
   from S3 S6 S7 show ( ( A · B ) · ( C · D ) ) = ( ( A · C ) · ( B · D
) )
     by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_muladd: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ and
    A4: D ∈ ℂ
   shows ( ( A + B ) · ( C + D ) ) =
  ( ( ( A · C ) + ( D · B ) ) + ( ( A · D ) + ( C · B ) ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI_pm3_2i)
   from A3 have S4: C ∈ ℂ.
   from A4 have S5: D ∈ ℂ.
   from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI_pm3_2i)
   have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
     ( ( A + B ) · ( C + D ) ) =
     ( ( ( A · C ) + ( D · B ) ) + ( ( A · D ) + ( C · B ) ) )
     by (rule MMI_muladdt)
   from S3 S6 S7 show
     ( ( A + B ) · ( C + D ) ) =
     ( ( ( A · C ) + ( D · B ) ) + ( ( A · D ) + ( C · B ) ) )
     by (rule MMI_mp2an)
qed


lemma (in MMIsar0) MMI_subdit:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
```

```
 ( A · ( B - C ) ) = ( ( A · B ) - ( A · C ) )
proof -
   have S1: ( A ∈ ℂ ∧ C ∈ ℂ ∧ ( B - C ) ∈ ℂ ) ⟶
 ( A · ( C + ( B - C ) ) ) =
     ( ( A · C ) + ( A · ( B - C ) ) ) by (rule MMI_axdistr)
   have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ A ∈ ℂ by (rule MMI_3simp1)
   have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ C ∈ ℂ by (rule MMI_3simp3)
   have S4: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B - C ) ∈ ℂ by (rule MMI_subclt)
   from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B - C ) ∈ ℂ
     by (rule MMI_3adant1)
   from S1 S2 S3 S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A · ( C + ( B - C ) ) ) =
     ( ( A · C ) + ( A · ( B - C ) ) ) by (rule MMI_syl3anc)
   have S7: ( C ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( C + ( B - C ) ) = B by (rule MMI_pncan3t)
   from S7 have S8: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( C + ( B - C ) ) = B by (rule
MMI_ancoms)
   from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( C + ( B - C ) )
= B by (rule MMI_3adant1)
   from S9 have S10: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A · ( C + ( B - C ) ) ) = ( A · B ) by (rule MMI_opreq2d)
   from S6 S10 have S11: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A · C ) + ( A · ( B - C ) ) ) = ( A · B ) by (rule MMI_eqtr3d)
   have S12: ( ( A · B ) ∈ ℂ ∧ ( A · C ) ∈ ℂ ∧ ( A · ( B - C ) ) ∈ ℂ
) ⟶
 ( ( ( A · B ) - ( A · C ) ) = ( A · ( B - C ) ) ⟷
     ( ( A · C ) + ( A · ( B - C ) ) ) = ( A · B ) ) by (rule MMI_subaddt)
   have S13: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) ∈ ℂ by (rule MMI_axmulcl)
   from S13 have S14: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · B ) ∈ ℂ
     by (rule MMI_3adant3)
   have S15: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · C ) ∈ ℂ by (rule MMI_axmulcl)
   from S15 have S16: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( A · C ) ∈ ℂ
     by (rule MMI_3adant2)
   have S17: ( A ∈ ℂ ∧ ( B - C ) ∈ ℂ ) ⟶ ( A · ( B - C ) ) ∈ ℂ
     by (rule MMI_axmulcl)
   from S4 have S18: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B - C ) ∈ ℂ .
   from S17 S18 have S19: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ C ∈ ℂ ) ) ⟶
     ( A · ( B - C ) ) ∈ ℂ by (rule MMI_sylan2)
   from S19 have S20: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
     ( A · ( B - C ) ) ∈ ℂ by (rule MMI_3impb)
   from S12 S14 S16 S20 have S21: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( ( A · B ) - ( A · C ) ) = ( A · ( B - C ) ) ⟷
     ( ( A · C ) + ( A · ( B - C ) ) ) = ( A · B ) ) by (rule MMI_syl3anc)
   from S11 S21 have S22: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A · B ) - ( A · C ) ) = ( A · ( B - C ) ) by (rule MMI_mpbird)
   from S22 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A · ( B - C ) ) = ( ( A · B ) - ( A · C ) ) by (rule MMI_eqcomd)
qed

lemma (in MMIsar0) MMI_subdirt:
```

shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = ((A · C) - (B · C))
proof -
have S1: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
(C · (A - B)) = ((C · A) - (C · B)) **by** (rule MMI_subdit)
from S1 **have** S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C · (A - B)) = ((C · A) - (C · B)) **by** (rule MMI_3coml)
have S3: ((A - B) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = (C · (A - B)) **by** (rule MMI_axmulcom)
have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ **by** (rule MMI_subclt)
from S3 S4 **have** S5: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A - B) · C) = (C · (A - B)) **by** (rule MMI_sylan)
from S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = (C · (A - B)) **by** (rule MMI_3impa)
have S7: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) = (C · A) **by** (rule MMI_axmulcom)
from S7 **have** S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) = (C ·
A)
by (rule MMI_3adant2)
have S9: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C · B) **by** (rule MMI_axmulcom)
from S9 **have** S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B · C) = (C ·
B)
by (rule MMI_3adant1)
from S8 S10 **have** S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · C) - (B · C)) = ((C · A) - (C · B))
by (rule MMI_opreq12d)
from S2 S6 S11 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) · C) = ((A · C) - (B · C)) **by** (rule MMI_3eqtr4d)
qed

lemma (**in** MMIsar0) MMI_subdi: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ **and**
A3: C ∈ ℂ
shows (A · (B - C)) = ((A · B) - (A · C))
proof -
from A1 **have** S1: A ∈ ℂ.
from A2 **have** S2: B ∈ ℂ.
from A3 **have** S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · (B - C)) = ((A · B) - (A · C)) **by** (rule MMI_subdit)
from S1 S2 S3 S4 **show** (A · (B - C)) = ((A · B) - (A · C))

by (rule MMI_mp3an)
qed

lemma (**in** MMIsar0) MMI_subdir: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ **and**
A3: C ∈ ℂ
shows ((A - B) · C) = ((A · C) - (B · C))
proof -

```
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: C ∈ ℂ.
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - B ) · C ) = ( ( A · C ) - ( B · C ) ) by (rule MMI_subdirt)
   from S1 S2 S3 S4 show ( ( A - B ) · C ) = ( ( A · C ) - ( B · C ) )

     by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_mul01: assumes A1: A ∈ ℂ
   shows ( A · 0 ) = 0
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: 0 ∈ ℂ by (rule MMI_0cn)
   have S3: 0 ∈ ℂ by (rule MMI_0cn)
   from S1 S2 S3 have S4: ( A · ( 0 - 0 ) ) = ( ( A · 0 ) - ( A · 0 )
)
     by (rule MMI_subdi)
   have S5: 0 ∈ ℂ by (rule MMI_0cn)
   from S5 have S6: ( 0 - 0 ) = 0 by (rule MMI_subid)
   from S6 have S7: ( A · ( 0 - 0 ) ) = ( A · 0 ) by (rule MMI_opreq2i)
   from A1 have S8: A ∈ ℂ.
   have S9: 0 ∈ ℂ by (rule MMI_0cn)
   from S8 S9 have S10: ( A · 0 ) ∈ ℂ by (rule MMI_mulcl)
   from S10 have S11: ( ( A · 0 ) - ( A · 0 ) ) = 0 by (rule MMI_subid)
   from S4 S7 S11 show ( A · 0 ) = 0 by (rule MMI_3eqtr3)
qed

lemma (in MMIsar0) MMI_mul02: assumes A1: A ∈ ℂ
   shows ( 0 · A ) = 0
proof -
   have S1: 0 ∈ ℂ by (rule MMI_0cn)
   from A1 have S2: A ∈ ℂ.
   from S1 S2 have S3: ( 0 · A ) = ( A · 0 ) by (rule MMI_mulcom)
   from A1 have S4: A ∈ ℂ.
   from S4 have S5: ( A · 0 ) = 0 by (rule MMI_mul01)
   from S3 S5 show ( 0 · A ) = 0 by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_1p1times: assumes A1: A ∈ ℂ
   shows ( ( 1 + 1 ) · A ) = ( A + A )
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   have S2: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S3: A ∈ ℂ.
   from S1 S2 S3 have S4: ( ( 1 + 1 ) · A ) = ( ( 1 · A ) + ( 1 · A )
)
     by (rule MMI_adddir)
```

```
    from A1 have S5: A ∈ ℂ.
    from S5 have S6: ( 1 · A ) = A by (rule MMI_mulid2)
    from S6 have S7: ( 1 · A ) = A .
    from S6 S7 have S8: ( ( 1 · A ) + ( 1 · A ) ) = ( A + A )
      by (rule MMI_opreq12i)
    from S4 S8 show ( ( 1 + 1 ) · A ) = ( A + A )
      by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_mul01t:
    shows A ∈ ℂ ⟶ ( A · 0 ) = 0
proof -
    have S1: A = if ( A ∈ ℂ , A , 0 ) ⟶
  ( A · 0 ) = ( if ( A ∈ ℂ , A , 0 ) · 0 ) by (rule MMI_opreq1)
    from S1 have S2: A = if ( A ∈ ℂ , A , 0 ) ⟶
  ( ( A · 0 ) = 0 ⟷ ( if ( A ∈ ℂ , A , 0 ) · 0 ) = 0 ) by (rule MMI_eqeq1d)
    have S3: 0 ∈ ℂ by (rule MMI_0cn)
    from S3 have S4: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
    from S4 have S5: ( if ( A ∈ ℂ , A , 0 ) · 0 ) = 0 by (rule MMI_mul01)
    from S2 S5 show A ∈ ℂ ⟶ ( A · 0 ) = 0 by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_mul02t:
    shows A ∈ ℂ ⟶ ( 0 · A ) = 0
proof -
    have S1: 0 ∈ ℂ by (rule MMI_0cn)
    have S2: ( 0 ∈ ℂ ∧ A ∈ ℂ ) ⟶ ( 0 · A ) = ( A · 0 ) by (rule MMI_axmulcom)
    from S1 S2 have S3: A ∈ ℂ ⟶ ( 0 · A ) = ( A · 0 ) by (rule MMI_mpan)
    have S4: A ∈ ℂ ⟶ ( A · 0 ) = 0 by (rule MMI_mul01t)
    from S3 S4 show A ∈ ℂ ⟶ ( 0 · A ) = 0 by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_mulneg1: assumes A1: A ∈ ℂ and
     A2: B ∈ ℂ
    shows ( ( (-  A) ) · B ) = ( - ( A · B ) )
proof -
    from A2 have S1: B ∈ ℂ.
    from S1 have S2: ( B · 0 ) = 0 by (rule MMI_mul01)
    from A2 have S3: B ∈ ℂ.
    from A1 have S4: A ∈ ℂ.
    from S3 S4 have S5: ( B · A ) = ( A · B ) by (rule MMI_mulcom)
    from S2 S5 have S6: ( ( B · 0 ) - ( B · A ) ) = ( 0 - ( A · B ) )
      by (rule MMI_opreq12i)
    have S7: ( (-  A) ) = ( 0 - A ) by (rule MMI_df_neg)
    from S7 have S8: ( ( (-  A) ) · B ) = ( ( 0 - A ) · B )
      by (rule MMI_opreq1i)
    have S9: 0 ∈ ℂ by (rule MMI_0cn)
    from A1 have S10: A ∈ ℂ.
    from S9 S10 have S11: ( 0 - A ) ∈ ℂ by (rule MMI_subcl)
```

```
   from A2 have S12: B ∈ ℂ.
   from S11 S12 have S13: ( ( 0 - A ) · B ) = ( B · ( 0 - A ) )
     by (rule MMI_mulcom)
   from A2 have S14: B ∈ ℂ.
   have S15: 0 ∈ ℂ by (rule MMI_0cn)
   from A1 have S16: A ∈ ℂ.
   from S14 S15 S16 have
     S17: ( B · ( 0 - A ) ) = ( ( B · 0 ) - ( B · A ) )
     by (rule MMI_subdi)
   from S8 S13 S17 have
     S18: ( ( (-  A) ) · B ) = ( ( B · 0 ) - ( B · A ) ) by (rule MMI_3eqtr)
   have S19: ( - ( A · B ) ) = ( 0 - ( A · B ) ) by (rule MMI_df_neg)
   from S6 S18 S19 show ( ( (-  A) ) · B ) = ( - ( A · B ) )
     by (rule MMI_3eqtr4)
qed
```

```
lemma (in MMIsar0) MMI_mulneg2: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( A · ( (-  B) ) ) =
 ( - ( A · B ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S2 have S3: ( (-  B) ) ∈ ℂ by (rule MMI_negcl)
   from S1 S3 have S4: ( A · ( (-  B) ) ) =
 ( ( (-  B) ) · A ) by (rule MMI_mulcom)
   from A2 have S5: B ∈ ℂ.
   from A1 have S6: A ∈ ℂ.
   from S5 S6 have S7: ( ( (-  B) ) · A ) =
 ( - ( B · A ) ) by (rule MMI_mulneg1)
   from A2 have S8: B ∈ ℂ.
   from A1 have S9: A ∈ ℂ.
   from S8 S9 have S10: ( B · A ) = ( A · B ) by (rule MMI_mulcom)
   from S10 have S11: ( - ( B · A ) ) =
 ( - ( A · B ) ) by (rule MMI_negeqi)
   from S4 S7 S11 show ( A · ( (-  B) ) ) =
 ( - ( A · B ) ) by (rule MMI_3eqtr)
qed
```

```
lemma (in MMIsar0) MMI_mul2neg: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( ( (-  A) ) · ( (-  B) ) ) =
 ( A · B )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S2 have S3: ( (-  B) ) ∈ ℂ by (rule MMI_negcl)
```

```
  from S1 S3 have S4: ( ( (-  A) ) · ( (-  B) ) ) =
 ( - ( A · ( (-  B) ) ) ) by (rule MMI_mulneg1)
  from A1 have S5: A ∈ ℂ.
  from S3 have S6: ( (-  B) ) ∈ ℂ .
  from S5 S6 have S7: ( A · ( (-  B) ) ) =
 ( ( (-  B) ) · A ) by (rule MMI_mulcom)
  from A2 have S8: B ∈ ℂ.
  from A1 have S9: A ∈ ℂ.
  from S8 S9 have S10: ( ( (-  B) ) · A ) =
 ( - ( B · A ) ) by (rule MMI_mulneg1)
  from S7 S10 have S11: ( A · ( (-  B) ) ) =
 ( - ( B · A ) ) by (rule MMI_eqtr)
  from S11 have S12: ( - ( A · ( (-  B) ) ) ) =
 ( - ( - ( B · A ) ) ) by (rule MMI_negeqi)
  from A2 have S13: B ∈ ℂ.
  from A1 have S14: A ∈ ℂ.
  from S13 S14 have S15: ( B · A ) ∈ ℂ by (rule MMI_mulcl)
  from S15 have S16: ( - ( - ( B · A ) ) ) =
 ( B · A ) by (rule MMI_negneg)
  from S4 S12 S16 have S17: ( ( (-  A) ) · ( (-  B) ) ) =
 ( B · A ) by (rule MMI_3eqtr)
  from A2 have S18: B ∈ ℂ.
  from A1 have S19: A ∈ ℂ.
  from S18 S19 have S20: ( B · A ) = ( A · B ) by (rule MMI_mulcom)
  from S17 S20 show ( ( (-  A) ) · ( (-  B) ) ) =
 ( A · B ) by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_negdi: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
  shows ( - ( A + B ) ) =
 ( ( (-  A) ) + ( (-  B) ) )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from S1 S2 have S3: ( A + B ) ∈ ℂ by (rule MMI_addcl)
  from S3 have S4: ( 1 · ( A + B ) ) =
 ( A + B ) by (rule MMI_mulid2)
  from S4 have S5: ( - ( 1 · ( A + B ) ) ) =
 ( - ( A + B ) ) by (rule MMI_negeqi)
  have S6: 1 ∈ ℂ by (rule MMI_1cn)
  from S6 have S7: ( - 1 ) ∈ ℂ by (rule MMI_negcl)
  from A1 have S8: A ∈ ℂ.
  from A2 have S9: B ∈ ℂ.
  from S7 S8 S9 have S10: ( ( - 1 ) · ( A + B ) ) =
 ( ( ( - 1 ) · A ) + ( ( - 1 ) · B ) ) by (rule MMI_adddi)
  have S11: 1 ∈ ℂ by (rule MMI_1cn)
  from S3 have S12: ( A + B ) ∈ ℂ .
  from S11 S12 have S13: ( ( - 1 ) · ( A + B ) ) =
```

```
 ( - ( 1 · ( A + B ) ) ) by (rule MMI_mulneg1)
   have S14: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S15: A ∈ ℂ.
   from S14 S15 have S16: ( ( - 1 ) · A ) =
 ( - ( 1 · A ) ) by (rule MMI_mulneg1)
   from A1 have S17: A ∈ ℂ.
   from S17 have S18: ( 1 · A ) = A by (rule MMI_mulid2)
   from S18 have S19: ( - ( 1 · A ) ) = ( (-  A) ) by (rule MMI_negeqi)
   from S16 S19 have S20: ( ( - 1 ) · A ) = ( (-  A) ) by (rule MMI_eqtr)
   have S21: 1 ∈ ℂ by (rule MMI_1cn)
   from A2 have S22: B ∈ ℂ.
   from S21 S22 have S23: ( ( - 1 ) · B ) =
 ( - ( 1 · B ) ) by (rule MMI_mulneg1)
   from A2 have S24: B ∈ ℂ.
   from S24 have S25: ( 1 · B ) = B by (rule MMI_mulid2)
   from S25 have S26: ( - ( 1 · B ) ) = ( (-  B) ) by (rule MMI_negeqi)
   from S23 S26 have S27: ( ( - 1 ) · B ) = ( (-  B) ) by (rule MMI_eqtr)
   from S20 S27 have S28: ( ( ( - 1 ) · A ) + ( ( - 1 ) · B ) ) =
 ( ( (-  A) ) + ( (-  B) ) ) by (rule MMI_opreq12i)
   from S10 S13 S28 have S29: ( - ( 1 · ( A + B ) ) ) =
 ( ( (-  A) ) + ( (-  B) ) ) by (rule MMI_3eqtr3)
   from S5 S29 show ( - ( A + B ) ) =
 ( ( (-  A) ) + ( (-  B) ) ) by (rule MMI_eqtr3)
qed

lemma (in MMIsar0) MMI_negsubdi: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows ( - ( A - B ) ) =
 ( ( (-  A) ) + B )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S2 have S3: ( (-  B) ) ∈ ℂ by (rule MMI_negcl)
   from S1 S3 have S4: ( - ( A + ( (-  B) ) ) ) =
 ( ( (-  A) ) + ( - ( (-  B) ) ) ) by (rule MMI_negdi)
   from A1 have S5: A ∈ ℂ.
   from A2 have S6: B ∈ ℂ.
   from S5 S6 have S7: ( A + ( (-  B) ) ) = ( A - B ) by (rule MMI_negsub)
   from S7 have S8: ( - ( A + ( (-  B) ) ) ) =
 ( - ( A - B ) ) by (rule MMI_negeqi)
   from A2 have S9: B ∈ ℂ.
   from S9 have S10: ( - ( (-  B) ) ) = B by (rule MMI_negneg)
   from S10 have S11: ( ( (-  A) ) + ( - ( (-  B) ) ) ) =
 ( ( (-  A) ) + B ) by (rule MMI_opreq2i)
   from S4 S8 S11 show ( - ( A - B ) ) =
 ( ( (-  A) ) + B ) by (rule MMI_3eqtr3)
qed

lemma (in MMIsar0) MMI_negsubdi2: assumes A1: A ∈ ℂ and
```

```
   A2: B ∈ ℂ
  shows ( - ( A - B ) ) = ( B - A )
proof -
  from A1 have S1: A ∈ ℂ.
  from A2 have S2: B ∈ ℂ.
  from S1 S2 have S3: ( - ( A - B ) ) =
 ( ( (-  A) ) + B ) by (rule MMI_negsubdi)
  from A1 have S4: A ∈ ℂ.
  from S4 have S5: ( (-  A) ) ∈ ℂ by (rule MMI_negcl)
  from A2 have S6: B ∈ ℂ.
  from S5 S6 have S7: ( ( (-  A) ) + B ) =
 ( B + ( (-  A) ) ) by (rule MMI_addcom)
  from A2 have S8: B ∈ ℂ.
  from A1 have S9: A ∈ ℂ.
  from S8 S9 have S10: ( B + ( (-  A) ) ) = ( B - A ) by (rule MMI_negsub)
  from S3 S7 S10 show ( - ( A - B ) ) = ( B - A ) by (rule MMI_3eqtr)
qed

lemma (in MMIsar0) MMI_mulneg1t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( (-  A) ) · B ) =
 ( - ( A · B ) )
proof -
  have S1: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( (-  A) ) =
 ( - if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_negeq)
  from S1 have S2: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( (-  A) ) · B ) =
 ( ( - if ( A ∈ ℂ , A , 0 ) ) · B ) by (rule MMI_opreq1d)
  have S3: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A · B ) =
 ( if ( A ∈ ℂ , A , 0 ) · B ) by (rule MMI_opreq1)
  from S3 have S4: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( - ( A · B ) ) =
 ( - ( if ( A ∈ ℂ , A , 0 ) · B ) ) by (rule MMI_negeqd)
  from S2 S4 have S5: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( ( (-  A) ) · B ) =
 ( - ( A · B ) ) ⟷
 ( ( - if ( A ∈ ℂ , A , 0 ) ) · B ) =
 ( - ( if ( A ∈ ℂ , A , 0 ) · B ) ) ) by (rule MMI_eqeq12d)
  have S6: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( - if ( A ∈ ℂ , A , 0 ) ) · B ) =
 ( ( - if ( A ∈ ℂ , A , 0 ) ) · if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
```

```
  have S7: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( if ( A ∈ ℂ , A , 0 ) · B ) =
 ( if ( A ∈ ℂ , A , 0 ) · if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
  from S7 have S8: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( - ( if ( A ∈ ℂ , A , 0 ) · B ) ) =
 ( - ( if ( A ∈ ℂ , A , 0 ) · if ( B ∈ ℂ , B , 0 ) ) ) by (rule MMI_negeqd)
  from S6 S8 have S9: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( ( - if ( A ∈ ℂ , A , 0 ) ) · B ) =
 ( - ( if ( A ∈ ℂ , A , 0 ) · B ) ) ⟷
 ( ( - if ( A ∈ ℂ , A , 0 ) ) · if ( B ∈ ℂ , B , 0 ) ) =
 ( - ( if ( A ∈ ℂ , A , 0 ) · if ( B ∈ ℂ , B , 0 ) ) ) ) by (rule MMI_eqeq12d)
  have S10: 0 ∈ ℂ by (rule MMI_0cn)
  from S10 have S11: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
  have S12: 0 ∈ ℂ by (rule MMI_0cn)
  from S12 have S13: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
  from S11 S13 have S14: ( ( - if ( A ∈ ℂ , A , 0 ) ) · if ( B ∈ ℂ ,
B , 0 ) ) =
 ( - ( if ( A ∈ ℂ , A , 0 ) · if ( B ∈ ℂ , B , 0 ) ) ) by (rule MMI_mulneg1)
  from S5 S9 S14 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( (-  A) ) · B ) =
 ( - ( A · B ) ) by (rule MMI_dedth2h)
qed

lemma (in MMIsar0) MMI_mulneg2t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( A · ( (-  B) ) ) =
 ( - ( A · B ) )
proof -
  have S1: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶
 ( ( (-  B) ) · A ) =
 ( - ( B · A ) ) by (rule MMI_mulneg1t)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( (-  B) ) · A ) =
 ( - ( B · A ) ) by (rule MMI_ancoms)
  have S3: ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ) ⟶
 ( A · ( (-  B) ) ) =
 ( ( (-  B) ) · A ) by (rule MMI_axmulcom)
  have S4: B ∈ ℂ ⟶ ( (-  B) ) ∈ ℂ by (rule MMI_negclt)
  from S3 S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( A · ( (-  B) ) ) =
 ( ( (-  B) ) · A ) by (rule MMI_sylan2)
  have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( A · B ) = ( B · A ) by (rule MMI_axmulcom)
  from S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( - ( A · B ) ) =
 ( - ( B · A ) ) by (rule MMI_negeqd)
```

from S2 S5 S7 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · ((- B))) =
(- (A · B)) **by** (rule MMI_3eqtr4d)
qed

lemma (**in** MMIsar0) MMI_mulneg12t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(A · ((- B)))
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(- (A · B)) **by** (rule MMI_mulneg1t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · ((- B))) =
(- (A · B)) **by** (rule MMI_mulneg2t)
from S1 S2 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · B) =
(A · ((- B))) **by** (rule MMI_eqtr4d)
qed

lemma (**in** MMIsar0) MMI_mul2negt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · ((- B))) =
(A · B)
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
((- A)) =
(- if (A ∈ ℂ , A , **0**)) **by** (rule MMI_negeq)
from S1 **have** S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(((- A)) · ((- B))) =
((- if (A ∈ ℂ , A , **0**)) · ((- B))) **by** (rule MMI_opreq1d)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · B) =
(if (A ∈ ℂ , A , **0**) · B) **by** (rule MMI_opreq1)
from S2 S3 **have** S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((((- A)) · ((- B))) =
(A · B) ⟷
((- if (A ∈ ℂ , A , **0**)) · ((- B))) =
(if (A ∈ ℂ , A , **0**) · B)) **by** (rule MMI_eqeq12d)
have S5: B =
if (B ∈ ℂ , B , **0**) ⟶
((- B)) =
(- if (B ∈ ℂ , B , **0**)) **by** (rule MMI_negeq)
from S5 **have** S6: B =

if (B ∈ ℂ , B , **0**) ⟶
((- if (A ∈ ℂ , A , **0**)) · ((- B))) =
((- if (A ∈ ℂ , A , **0**)) · (- if (B ∈ ℂ , B , **0**))) **by** (rule MMI_opreq2d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · B) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) **by** (rule MMI_opreq2)
from S6 S7 **have** S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(((- if (A ∈ ℂ , A , **0**)) · ((- B))) =
(if (A ∈ ℂ , A , **0**) · B) ⟷
((- if (A ∈ ℂ , A , **0**)) · (- if (B ∈ ℂ , B , **0**))) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**))) **by** (rule MMI_eqeq12d)
have S9: **0** ∈ ℂ **by** (rule MMI_0cn)
from S9 **have** S10: if (A ∈ ℂ , A , **0**) ∈ ℂ **by** (rule MMI_elimel)
have S11: **0** ∈ ℂ **by** (rule MMI_0cn)
from S11 **have** S12: if (B ∈ ℂ , B , **0**) ∈ ℂ **by** (rule MMI_elimel)
from S10 S12 **have** S13: ((- if (A ∈ ℂ , A , **0**)) · (- if (B ∈ ℂ , B , **0**))) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) **by** (rule MMI_mul2neg)
from S4 S8 S13 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(((- A)) · ((- B))) =
(A · B) **by** (rule MMI_dedth2h)
qed

lemma (**in** MMIsar0) MMI_negdit:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) =
(((- A)) + ((- B)))
proof -
have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A + B) =
(if (A ∈ ℂ , A , **0**) + B) **by** (rule MMI_opreq1)
from S1 **have** S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(- (A + B)) =
(- (if (A ∈ ℂ , A , **0**) + B)) **by** (rule MMI_negeqd)
have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
((- A)) =
(- if (A ∈ ℂ , A , **0**)) **by** (rule MMI_negeq)
from S3 **have** S4: A =
if (A ∈ ℂ , A , **0**) ⟶
(((- A)) + ((- B))) =
((- if (A ∈ ℂ , A , **0**)) + ((- B))) **by** (rule MMI_opreq1d)
from S2 S4 **have** S5: A =
if (A ∈ ℂ , A , **0**) ⟶

((- (A + B)) =
(((- A)) + ((- B))) ⟷
(- (if (A ∈ ℂ , A , **0**) + B)) =
((- if (A ∈ ℂ , A , **0**)) + ((- B)))) **by** (rule MMI_eqeq12d)
have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) + B) =
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) **by** (rule MMI_opreq2)
from S6 **have** S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(- (if (A ∈ ℂ , A , **0**) + B)) =
(- (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**))) **by** (rule MMI_negeqd)
have S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((- B)) =
(- if (B ∈ ℂ , B , **0**)) **by** (rule MMI_negeq)
from S8 **have** S9: B =
if (B ∈ ℂ , B , **0**) ⟶
((- if (A ∈ ℂ , A , **0**)) + ((- B))) =
((- if (A ∈ ℂ , A , **0**)) + (- if (B ∈ ℂ , B , **0**))) **by** (rule
MMI_opreq2d)
from S7 S9 **have** S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((- (if (A ∈ ℂ , A , **0**) + B)) =
((- if (A ∈ ℂ , A , **0**)) + ((- B))) ⟷
(- (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**))) =
((- if (A ∈ ℂ , A , **0**)) + (- if (B ∈ ℂ , B , **0**)))) **by** (rule
MMI_eqeq12d)
have S11: **0** ∈ ℂ **by** (rule MMI_0cn)
from S11 **have** S12: if (A ∈ ℂ , A , **0**) ∈ ℂ **by** (rule MMI_elimel)
have S13: **0** ∈ ℂ **by** (rule MMI_0cn)
from S13 **have** S14: if (B ∈ ℂ , B , **0**) ∈ ℂ **by** (rule MMI_elimel)
from S12 S14 **have** S15: (- (if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ ,
B , **0**))) =
((- if (A ∈ ℂ , A , **0**)) + (- if (B ∈ ℂ , B , **0**))) **by** (rule
MMI_negdi)
from S5 S10 S15 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) =
(((- A)) + ((- B))) **by** (rule MMI_dedth2h)
qed

lemma (**in** MMIsar0) MMI_negdi2t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) = (((- A)) - B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(- (A + B)) =

```
( ( (-  A) ) + ( (-  B) ) ) by (rule MMI_negdit)
   have S2: ( ( (-  A) ) ∈ ℂ ∧ B ∈ ℂ ) ⟶
( ( (-  A) ) + ( (-  B) ) ) =
( ( (-  A) ) - B ) by (rule MMI_negsubt)
   have S3: A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI_negclt)
   from S2 S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( ( (-  A) ) + ( (-  B) ) ) =
( ( (-  A) ) - B ) by (rule MMI_sylan)
   from S1 S4 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A + B ) ) = ( ( (-  A) ) - B )
 by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_negsubdit:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A - B ) ) = ( ( (-  A) ) + B )
proof -
   have S1: ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ) ⟶
( - ( A + ( (-  B) ) ) ) =
( ( (-  A) ) + ( - ( (-  B) ) ) ) by (rule MMI_negdit)
   have S2: B ∈ ℂ ⟶ ( (-  B) ) ∈ ℂ by (rule MMI_negclt)
   from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A + ( (-  B) ) ) ) =
( ( (-  A) ) + ( - ( (-  B) ) ) ) by (rule MMI_sylan2)
   have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( A + ( (-  B) ) ) = ( A - B ) by (rule MMI_negsubt)
   from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A + ( (-  B) ) ) ) =
( - ( A - B ) ) by (rule MMI_negeqd)
   have S6: B ∈ ℂ ⟶ ( - ( (-  B) ) ) = B by (rule MMI_negnegt)
   from S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( - ( (-  B) ) ) = B
     by (rule MMI_adantl)
   from S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( ( (-  A) ) + ( - ( (-  B) ) ) ) =
( ( (-  A) ) + B ) by (rule MMI_opreq2d)
   from S3 S5 S8 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A - B ) ) = ( ( (-  A) ) + B )
 by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_negsubdi2t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A - B ) ) = ( B - A )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
( - ( A - B ) ) = ( ( (-  A) ) + B ) by (rule MMI_negsubdit)
   have S2: ( ( (-  A) ) ∈ ℂ ∧ B ∈ ℂ ) ⟶
( ( (-  A) ) + B ) = ( B + ( (-  A) ) ) by (rule MMI_axaddcom)
   have S3: A ∈ ℂ ⟶ ( (-  A) ) ∈ ℂ by (rule MMI_negclt)
```

```
  from S2 S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( (-  A) ) + B ) = ( B + ( (-  A) ) ) by (rule MMI_sylan)
  have S5: ( B ∈ ℂ ∧ A ∈ ℂ ) ⟶
 ( B + ( (-  A) ) ) = ( B - A ) by (rule MMI_negsubt)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( B + ( (-  A) ) ) = ( B - A ) by (rule MMI_ancoms)
  from S1 S4 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( - ( A - B ) ) = ( B - A )
 by (rule MMI_3eqtrd)
qed

lemma (in MMIsar0) MMI_subsub2t:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( A + ( C - B ) )
proof -
  have S1: ( A ∈ ℂ ∧ ( B - C ) ∈ ℂ ) ⟶
 ( A + ( - ( B - C ) ) ) =
 ( A - ( B - C ) ) by (rule MMI_negsubt)
  have S2: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B - C ) ∈ ℂ by (rule MMI_subclt)
  from S1 S2 have S3: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ C ∈ ℂ ) ) ⟶
 ( A + ( - ( B - C ) ) ) =
 ( A - ( B - C ) ) by (rule MMI_sylan2)
  from S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + ( - ( B - C ) ) ) =
 ( A - ( B - C ) ) by (rule MMI_3impb)
  have S5: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( - ( B - C ) ) = ( C - B ) by (rule MMI_negsubdi2t)
  from S5 have S6: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + ( - ( B - C ) ) ) =
 ( A + ( C - B ) ) by (rule MMI_opreq2d)
  from S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + ( - ( B - C ) ) ) =
 ( A + ( C - B ) ) by (rule MMI_3adant1)
  from S4 S7 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( A + ( C - B ) )
 by (rule MMI_eqtr3d)
qed

lemma (in MMIsar0) MMI_subsubt:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( ( A - B ) + C )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( A + ( C - B ) ) by (rule MMI_subsub2t)
  have S2: ( A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( A + C ) - B ) = ( A + ( C - B ) ) by (rule MMI_addsubasst)
  have S3: ( A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( A + C ) - B ) = ( ( A - B ) + C ) by (rule MMI_addsubt)
  from S2 S3 have S4: ( A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶
```

```
 ( A + ( C - B ) ) = ( ( A - B ) + C ) by (rule MMI_eqtr3d)
   from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + ( C - B ) ) = ( ( A - B ) + C ) by (rule MMI_3com23)
   from S1 S5 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( ( A - B ) + C )
 by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_subsub3t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( ( A + C ) - B )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( A + ( C - B ) ) by (rule MMI_subsub2t)
   have S2: ( A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( A + C ) - B ) = ( A + ( C - B ) ) by (rule MMI_addsubasst)
   from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + C ) - B ) = ( A + ( C - B ) ) by (rule MMI_3com23)
   from S1 S3 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - C ) ) = ( ( A + C ) - B )
 by (rule MMI_eqtr4d)
qed

lemma (in MMIsar0) MMI_subsub4t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - B ) - C ) = ( A - ( B + C ) )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( - C ) ∈ ℂ ) ⟶
 ( A - ( B - ( - C ) ) ) =
 ( ( A - B ) + ( - C ) ) by (rule MMI_subsubt)
   have S2: C ∈ ℂ ⟶ ( - C ) ∈ ℂ by (rule MMI_negclt)
   from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - ( - C ) ) ) =
 ( ( A - B ) + ( - C ) ) by (rule MMI_syl3an3)
   have S4: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B - ( - C ) ) = ( B + C ) by (rule MMI_subnegt)
   from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B - ( - C ) ) = ( B + C ) by (rule MMI_3adant1)
   from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( B - ( - C ) ) ) =
 ( A - ( B + C ) ) by (rule MMI_opreq2d)
   have S7: ( ( A - B ) ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - B ) + ( - C ) ) =
 ( ( A - B ) - C ) by (rule MMI_negsubt)
   have S8: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A - B ) ∈ ℂ by (rule MMI_subclt)
   from S7 S8 have S9: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ C ∈ ℂ ) ⟶
 ( ( A - B ) + ( - C ) ) =
 ( ( A - B ) - C ) by (rule MMI_sylan)
   from S9 have S10: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
```

((A - B) + (- C)) =
((A - B) - C) **by** (rule MMI_3impa)
from S3 S6 S10 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = (A - (B + C))
by (rule MMI_3eqtr3rd)
qed

lemma **(in** MMIsar0) MMI_sub23t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = ((A - C) - B)
proof -
have S1: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) = (C + B) **by** (rule MMI_axaddcom)
from S1 **have** S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B + C) = (C + B) **by** (rule MMI_3adant1)
from S2 **have** S3: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - (B + C)) = (A - (C + B)) **by** (rule MMI_opreq2d)
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = (A - (B + C)) **by** (rule MMI_subsub4t)
have S5: (A ∈ ℂ ∧ C ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - C) - B) = (A - (C + B)) **by** (rule MMI_subsub4t)
from S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) - B) = (A - (C + B)) **by** (rule MMI_3com23)
from S3 S4 S6 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - C) = ((A - C) - B)
by (rule MMI_3eqtr4d)
qed

lemma **(in** MMIsar0) MMI_nnncant:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ (B - C) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
(A - ((B - C) + C)) **by** (rule MMI_subsub4t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ **by** (rule MMI_3simp1)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ **by** (rule MMI_subclt)
from S3 **have** S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - C) ∈ ℂ **by** (rule MMI_3adant1)
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ **by** (rule MMI_3simp3)
from S1 S2 S4 S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
(A - ((B - C) + C)) **by** (rule MMI_syl3anc)
have S7: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B - C) + C) = B **by** (rule MMI_npcant)
from S7 **have** S8: (B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B - C) + C)) = (A - B) **by** (rule MMI_opreq2d)
from S8 **have** S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - ((B - C) + C)) = (A - B) **by** (rule MMI_3adant1)

from S6 S9 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) = (A - B)
by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_nnncan1t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - (A - C)) = (C - B)
proof -
have S1: ((A - B) ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶
((A - B) + (- (A - C))) =
((A - B) - (A - C)) **by** (rule MMI_negsubt)
have S2: ((A - B) ∈ ℂ ∧ (- (A - C)) ∈ ℂ) ⟶
((A - B) + (- (A - C))) =
((- (A - C)) + (A - B)) **by** (rule MMI_axaddcom)
have S3: (A - C) ∈ ℂ ⟶ (- (A - C)) ∈ ℂ
by (rule MMI_negclt)
from S2 S3 **have** S4: ((A - B) ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶
((A - B) + (- (A - C))) =
((- (A - C)) + (A - B)) **by** (rule MMI_sylan2)
from S1 S4 **have** S5: ((A - B) ∈ ℂ ∧ (A - C) ∈ ℂ) ⟶
((A - B) - (A - C)) =
((- (A - C)) + (A - B)) **by** (rule MMI_eqtr3d)
have S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ **by** (rule MMI_subclt)
from S6 **have** S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - B) ∈ ℂ **by** (rule MMI_3adant3)
have S8: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A - C) ∈ ℂ **by** (rule MMI_subclt)
from S8 **have** S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A - C) ∈ ℂ **by** (rule MMI_3adant2)
from S5 S7 S9 **have** S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - (A - C)) =
((- (A - C)) + (A - B)) **by** (rule MMI_sylanc)
have S11: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(- (A - C)) = (C - A) **by** (rule MMI_negsubdi2t)
from S11 **have** S12: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(- (A - C)) = (C - A) **by** (rule MMI_3adant2)
from S12 **have** S13: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((- (A - C)) + (A - B)) =
((C - A) + (A - B)) **by** (rule MMI_opreq1d)
have S14: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
((C - A) + (A - B)) = (C - B) **by** (rule MMI_npncant)
from S14 **have** S15: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((C - A) + (A - B)) = (C - B) **by** (rule MMI_3coml)
from S10 S13 S15 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - B) - (A - C)) = (C - B)
by (rule MMI_3eqtrd)
qed

lemma (**in** MMIsar0) MMI_nnncan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) - (B - C)) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ (B - C) ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
((A - C) - (B - C)) **by** (rule MMI_sub23t)
have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ **by** (rule MMI_3simp1)
have S3: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (B - C) ∈ ℂ **by** (rule MMI_subclt)
from S3 **have** S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B - C) ∈ ℂ **by** (rule MMI_3adant1)
have S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ **by** (rule MMI_3simp3)
from S1 S2 S4 S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) =
((A - C) - (B - C)) **by** (rule MMI_syl3anc)
have S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B - C)) - C) = (A - B) **by** (rule MMI_nnncant)
from S6 S7 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - C) - (B - C)) = (A - B) **by** (rule MMI_eqtr3d)
qed

lemma (**in** MMIsar0) MMI_nncant:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A - (A - B)) = B
proof -
have S1: **0** ∈ ℂ **by** (rule MMI_0cn)
have S2: (A ∈ ℂ ∧ **0** ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - **0**) - (A - B)) = (B - **0**) **by** (rule MMI_nnncan1t)
from S1 S2 **have** S3: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - **0**) - (A - B)) = (B - **0**) **by** (rule MMI_mp3an2)
have S4: A ∈ ℂ ⟶ (A - **0**) = A **by** (rule MMI_subid1t)
from S4 **have** S5: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - **0**) = A
by (rule MMI_adantr)
from S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A - **0**) - (A - B)) =
(A - (A - B)) **by** (rule MMI_opreq1d)
have S7: B ∈ ℂ ⟶ (B - **0**) = B **by** (rule MMI_subid1t)
from S7 **have** S8: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (B - **0**) = B
by (rule MMI_adantl)
from S3 S6 S8 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A - (A - B)) = B **by** (rule MMI_3eqtr3d)
qed

lemma (**in** MMIsar0) MMI_nppcan2t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A - (B + C)) + C) = (A - B)
proof -
have S1: (A ∈ ℂ ∧ (B + C) ∈ ℂ ∧ C ∈ ℂ) ⟶

```
 ( A - ( ( B + C ) - C ) ) =
 ( ( A - ( B + C ) ) + C ) by (rule MMI_subsubt)
   have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ A ∈ ℂ by (rule MMI_3simp1)
   have S3: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B + C ) ∈ ℂ by (rule MMI_axaddcl)
   from S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B + C ) ∈ ℂ by (rule MMI_3adant1)
   have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ C ∈ ℂ by (rule MMI_3simp3)
   from S1 S2 S4 S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( ( B + C ) - C ) ) =
 ( ( A - ( B + C ) ) + C ) by (rule MMI_syl3anc)
   have S7: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( B + C ) - C ) = B by (rule MMI_pncant)
   from S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( B + C ) - C ) = B by (rule MMI_3adant1)
   from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A - ( ( B + C ) - C ) ) = ( A - B ) by (rule MMI_opreq2d)
   from S6 S9 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A - ( B + C ) ) + C ) = ( A - B ) by (rule MMI_eqtr3d)
qed

lemma (in MMIsar0) MMI_mulm1t:
   shows A ∈ ℂ ⟶ ( ( - 1 ) · A ) = ( (-  A) )
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   have S2: ( 1 ∈ ℂ ∧ A ∈ ℂ ) ⟶
 ( ( - 1 ) · A ) = ( - ( 1 · A ) ) by (rule MMI_mulneg1t)
   from S1 S2 have S3: A ∈ ℂ ⟶
 ( ( - 1 ) · A ) = ( - ( 1 · A ) ) by (rule MMI_mpan)
   have S4: A ∈ ℂ ⟶ ( 1 · A ) = A by (rule MMI_mulid2t)
   from S4 have S5: A ∈ ℂ ⟶ ( - ( 1 · A ) ) = ( (-  A) )
     by (rule MMI_negeqd)
   from S3 S5 show A ∈ ℂ ⟶ ( ( - 1 ) · A ) = ( (-  A) )
     by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_mulm1: assumes A1: A ∈ ℂ
   shows ( ( - 1 ) · A ) = ( (-  A) )
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: A ∈ ℂ ⟶ ( ( - 1 ) · A ) = ( (-  A) ) by (rule MMI_mulm1t)
   from S1 S2 show ( ( - 1 ) · A ) = ( (-  A) ) by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_sub4t:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
 ( ( A + B ) - ( C + D ) ) =
 ( ( A - C ) + ( B - D ) )
proof -
   have S1: ( C ∈ ℂ ∧ D ∈ ℂ ) ⟶
```

(- (C + D)) =
((- C) + (- D)) **by** (rule MMI_negdit)
  from S1 **have** S2: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(- (C + D)) =
((- C) + (- D)) **by** (rule MMI_adantl)
  from S2 **have** S3: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A + B) + (- (C + D))) =
((A + B) + ((- C) + (- D)))
    by (rule MMI_opreq2d)
  have S4:
    ((A ∈ ℂ ∧ B ∈ ℂ) ∧ ((- C) ∈ ℂ ∧ (- D) ∈ ℂ)) ⟶
((A + B) + ((- C) + (- D))) =
((A + (- C)) + (B + (- D))) **by** (rule MMI_add4t)
  have S5: C ∈ ℂ ⟶ (- C) ∈ ℂ **by** (rule MMI_negclt)
  have S6: D ∈ ℂ ⟶ (- D) ∈ ℂ **by** (rule MMI_negclt)
  from S5 S6 **have** S7: (C ∈ ℂ ∧ D ∈ ℂ) ⟶
((- C) ∈ ℂ ∧ (- D) ∈ ℂ) **by** (rule MMI_anim12i)
  from S4 S7 **have** S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A + B) + ((- C) + (- D))) =
((A + (- C)) + (B + (- D))) **by** (rule MMI_sylan2)
  from S3 S8 **have** S9: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A + B) + (- (C + D))) =
((A + (- C)) + (B + (- D))) **by** (rule MMI_eqtrd)
  have S10: ((A + B) ∈ ℂ ∧ (C + D) ∈ ℂ) ⟶
((A + B) + (- (C + D))) =
((A + B) - (C + D)) **by** (rule MMI_negsubt)
  have S11: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + B) ∈ ℂ **by** (rule MMI_axaddcl)
  have S12: (C ∈ ℂ ∧ D ∈ ℂ) ⟶ (C + D) ∈ ℂ **by** (rule MMI_axaddcl)
  from S10 S11 S12 **have** S13:
    ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + B) + (- (C + D))) =
((A + B) - (C + D)) **by** (rule MMI_syl2an)
  have S14: (A ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- C)) = (A - C) **by** (rule MMI_negsubt)
  from S14 **have** S15: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(A + (- C)) = (A - C) **by** (rule MMI_ad2ant2r)
  have S16: (B ∈ ℂ ∧ D ∈ ℂ) ⟶
(B + (- D)) = (B - D) **by** (rule MMI_negsubt)
  from S16 **have** S17: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

(B + (- D)) = (B - D) **by** (rule MMI_ad2ant2l)
  from S15 S17 **have** S18: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)
) ⟶
((A + (- C)) + (B + (- D))) =

```
 ( ( A - C ) + ( B - D ) ) by (rule MMI_opreq12d)
   from S9 S13 S18 show ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) )
⟶
 ( ( A + B ) - ( C + D ) ) =
 ( ( A - C ) + ( B - D ) ) by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_sub4: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ and
    A4: D ∈ ℂ
   shows ( ( A + B ) - ( C + D ) ) =
 ( ( A - C ) + ( B - D ) )
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from S1 S2 have S3: A ∈ ℂ ∧ B ∈ ℂ by (rule MMI_pm3_2i)
   from A3 have S4: C ∈ ℂ.
   from A4 have S5: D ∈ ℂ.
   from S4 S5 have S6: C ∈ ℂ ∧ D ∈ ℂ by (rule MMI_pm3_2i)
   have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
 ( ( A + B ) - ( C + D ) ) =
 ( ( A - C ) + ( B - D ) ) by (rule MMI_sub4t)
   from S3 S6 S7 show ( ( A + B ) - ( C + D ) ) =
 ( ( A - C ) + ( B - D ) ) by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_mulsubt:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶
 ( ( A - B ) · ( C - D ) ) =
 ( ( ( A · C ) + ( D · B ) ) - ( ( A · D ) + ( C · B ) ) )
proof -
   have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( A + ( (-  B) ) ) = ( A - B ) by (rule MMI_negsubt)
   have S2: ( C ∈ ℂ ∧ D ∈ ℂ ) ⟶
 ( C + ( - D ) ) = ( C - D ) by (rule MMI_negsubt)
   from S1 S2 have S3: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈ ℂ ) ) ⟶

 ( ( A + ( (-  B) ) ) · ( C + ( - D ) ) ) =
 ( ( A - B ) · ( C - D ) ) by (rule MMI_opreqan12d)
   have S4: ( ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ) ∧ ( C ∈ ℂ ∧ ( - D ) ∈ ℂ )
) ⟶
 ( ( A + ( (-  B) ) ) · ( C + ( - D ) ) ) =
 ( ( ( A · C ) + ( ( - D ) · ( (-  B) ) ) ) + ( ( A · ( - D ) ) + ( C ·
( (-  B) ) ) ) ) by (rule MMI_muladdt)
   have S5: D ∈ ℂ ⟶ ( - D ) ∈ ℂ by (rule MMI_negclt)
   from S4 S5 have S6: ( ( A ∈ ℂ ∧ ( (-  B) ) ∈ ℂ ) ∧ ( C ∈ ℂ ∧ D ∈
ℂ ) ) ⟶
 ( ( A + ( (-  B) ) ) · ( C + ( - D ) ) ) =
```

(((A · C) + ((- D) · ((- B)))) +
((A · (- D)) + (C · ((- B)))))) **by** (rule MMI_sylanr2)
have S7: B ∈ ℂ ⟶ ((- B)) ∈ ℂ **by** (rule MMI_negclt)
from S6 S7 **have** S8: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A + ((- B))) · (C + (- D))) =
(((A · C) + ((- D) · ((- B))))
+ ((A · (- D)) + (C · ((- B))))))
by (rule MMI_sylanl2)
have S9: (D ∈ ℂ ∧ B ∈ ℂ) ⟶
((- D) · ((- B))) = (D · B) **by** (rule MMI_mul2negt)
from S9 **have** S10: (B ∈ ℂ ∧ D ∈ ℂ) ⟶
((- D) · ((- B))) = (D · B) **by** (rule MMI_ancoms)
from S10 **have** S11: (B ∈ ℂ ∧ D ∈ ℂ) ⟶
((A · C) + ((- D) · ((- B)))) =
((A · C) + (D · B)) **by** (rule MMI_opreq2d)
from S11 **have** S12: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A · C) + ((- D) · ((- B)))) =
((A · C) + (D · B)) **by** (rule MMI_ad2ant2l)
have S13: (A ∈ ℂ ∧ D ∈ ℂ) ⟶
(A · (- D)) = (- (A · D)) **by** (rule MMI_mulneg2t)
have S14: (C ∈ ℂ ∧ B ∈ ℂ) ⟶
(C · ((- B))) = (- (C · B)) **by** (rule MMI_mulneg2t)
from S13 S14 **have** S15: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (C ∈ ℂ ∧ B ∈ ℂ)
) ⟶
((A · (- D)) + (C · ((- B)))) =
((- (A · D)) + (- (C · B))) **by** (rule MMI_opreqan12d)
have S16: ((A · D) ∈ ℂ ∧ (C · B) ∈ ℂ) ⟶
(- ((A · D) + (C · B))) =
((- (A · D)) + (- (C · B))) **by** (rule MMI_negdit)
have S17: (A ∈ ℂ ∧ D ∈ ℂ) ⟶ (A · D) ∈ ℂ **by** (rule MMI_axmulcl)
have S18: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C · B) ∈ ℂ **by** (rule MMI_axmulcl)
from S16 S17 S18 **have** S19:
((A ∈ ℂ ∧ D ∈ ℂ) ∧ (C ∈ ℂ ∧ B ∈ ℂ)) ⟶
(- ((A · D) + (C · B))) =
((- (A · D)) + (- (C · B))) **by** (rule MMI_syl2an)
from S15 S19 **have** S20: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (C ∈ ℂ ∧ B ∈ ℂ)
) ⟶
((A · (- D)) + (C · ((- B)))) =
(- ((A · D) + (C · B))) **by** (rule MMI_eqtr4d)
from S20 **have** S21: ((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶

((A · (- D)) + (C · ((- B)))) =
(- ((A · D) + (C · B))) **by** (rule MMI_ancom2s)
from S21 **have** S22: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A · (- D)) + (C · ((- B)))) =
(- ((A · D) + (C · B))) **by** (rule MMI_an42s)

from S12 S22 **have** S23: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) + ((- D) · ((- B)))) +
((A · (- D)) + (C · ((- B))))) =
(((A · C) + (D · B)) + (- ((A · D) +
(C · B)))) **by** (rule MMI_opreq12d)
have S24: (((A · C) + (D · B)) ∈ ℂ ∧ ((A · D) +
(C · B)) ∈ ℂ) ⟶
(((A · C) + (D · B)) + (- ((A · D) + (C · B)))) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
by (rule MMI_negsubt)
have S25: ((A · C) ∈ ℂ ∧ (D · B) ∈ ℂ) ⟶
((A · C) + (D · B)) ∈ ℂ **by** (rule MMI_axaddcl)
have S26: (A ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · C) ∈ ℂ **by** (rule MMI_axmulcl)
have S27: (D ∈ ℂ ∧ B ∈ ℂ) ⟶ (D · B) ∈ ℂ **by** (rule MMI_axmulcl)
from S27 **have** S28: (B ∈ ℂ ∧ D ∈ ℂ) ⟶ (D · B) ∈ ℂ
by (rule MMI_ancoms)
from S25 S26 S28 **have** S29:
((A ∈ ℂ ∧ C ∈ ℂ) ∧ (B ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · C) + (D · B)) ∈ ℂ **by** (rule MMI_syl2an)
from S29 **have** S30: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A · C) + (D · B)) ∈ ℂ **by** (rule MMI_an4s)
have S31: ((A · D) ∈ ℂ ∧ (C · B) ∈ ℂ) ⟶
((A · D) + (C · B)) ∈ ℂ **by** (rule MMI_axaddcl)
from S17 **have** S32: (A ∈ ℂ ∧ D ∈ ℂ) ⟶ (A · D) ∈ ℂ .
from S18 **have** S33: (C ∈ ℂ ∧ B ∈ ℂ) ⟶ (C · B) ∈ ℂ .
from S33 **have** S34: (B ∈ ℂ ∧ C ∈ ℂ) ⟶ (C · B) ∈ ℂ
by (rule MMI_ancoms)
from S31 S32 S34 **have** S35:
((A ∈ ℂ ∧ D ∈ ℂ) ∧ (B ∈ ℂ ∧ C ∈ ℂ)) ⟶
((A · D) + (C · B)) ∈ ℂ **by** (rule MMI_syl2an)
from S35 **have** S36: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A · D) + (C · B)) ∈ ℂ **by** (rule MMI_an42s)
from S24 S30 S36 **have** S37:
((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
(((A · C) + (D · B)) + (- ((A · D) + (C · B)))) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
by (rule MMI_sylanc)
from S8 S23 S37 **have** S38: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A + ((- B))) · (C + (- D))) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))
by (rule MMI_3eqtrd)
from S3 S38 **show** ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶

((A - B) · (C - D)) =
(((A · C) + (D · B)) - ((A · D) + (C · B)))

```
    by (rule MMI_eqtr3d)
qed

lemma (in MMIsar0) MMI_pnpcant:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( A + C ) ) = ( B - C )
proof -
   have S1: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ ( A ∈ ℂ ∧ C ∈ ℂ ) ) ⟶
 ( ( A + B ) - ( A + C ) ) =
 ( ( A - A ) + ( B - C ) ) by (rule MMI_sub4t)
   from S1 have S2: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ C ∈ ℂ ) ) ⟶
 ( ( A + B ) - ( A + C ) ) =
 ( ( A - A ) + ( B - C ) ) by (rule MMI_anandis)
   have S3: A ∈ ℂ ⟶ ( A - A ) = 0 by (rule MMI_subidt)
   from S3 have S4: A ∈ ℂ ⟶
 ( ( A - A ) + ( B - C ) ) =
 ( 0 + ( B - C ) ) by (rule MMI_opreq1d)
   have S5: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶ ( B - C ) ∈ ℂ by (rule MMI_subclt)
   have S6: ( B - C ) ∈ ℂ ⟶
 ( 0 + ( B - C ) ) = ( B - C ) by (rule MMI_addid2t)
   from S5 S6 have S7: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( 0 + ( B - C ) ) = ( B - C ) by (rule MMI_syl)
   from S4 S7 have S8: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ C ∈ ℂ ) ) ⟶
 ( ( A - A ) + ( B - C ) ) = ( B - C ) by (rule MMI_sylan9eq)
   from S2 S8 have S9: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ C ∈ ℂ ) ) ⟶
 ( ( A + B ) - ( A + C ) ) = ( B - C ) by (rule MMI_eqtrd)
   from S9 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( A + C ) ) = ( B - C ) by (rule MMI_3impb)
qed

lemma (in MMIsar0) MMI_pnpcan2t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + C ) - ( B + C ) ) = ( A - B )
proof -
   have S1: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + C ) = ( C + A ) by (rule MMI_axaddcom)
   from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + C ) = ( C + A ) by (rule MMI_3adant2)
   have S3: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B + C ) = ( C + B ) by (rule MMI_axaddcom)
   from S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B + C ) = ( C + B ) by (rule MMI_3adant1)
   from S2 S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + C ) - ( B + C ) ) =
 ( ( C + A ) - ( C + B ) ) by (rule MMI_opreq12d)
   have S6: ( C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( C + A ) - ( C + B ) ) = ( A - B ) by (rule MMI_pnpcant)
   from S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( C + A ) - ( C + B ) ) = ( A - B ) by (rule MMI_3coml)
```

```
  from S5 S7 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + C ) - ( B + C ) ) = ( A - B ) by (rule MMI_eqtrd)
qed


lemma (in MMIsar0) MMI_pnncant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( A - C ) ) = ( B + C )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( - C ) ∈ ℂ ) ⟶
 ( ( A + B ) - ( A + ( - C ) ) ) =
 ( B - ( - C ) ) by (rule MMI_pnpcant)
  have S2: C ∈ ℂ ⟶ ( - C ) ∈ ℂ by (rule MMI_negclt)
  from S1 S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( A + ( - C ) ) ) =
 ( B - ( - C ) ) by (rule MMI_syl3an3)
  have S4: ( A ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + ( - C ) ) = ( A - C ) by (rule MMI_negsubt)
  from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + ( - C ) ) = ( A - C ) by (rule MMI_3adant2)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( A + ( - C ) ) ) =
 ( ( A + B ) - ( A - C ) ) by (rule MMI_opreq2d)
  have S7: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B - ( - C ) ) = ( B + C ) by (rule MMI_subnegt)
  from S7 have S8: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B - ( - C ) ) = ( B + C ) by (rule MMI_3adant1)
  from S3 S6 S8 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( A - C ) ) = ( B + C ) by (rule MMI_3eqtr3d)
qed

lemma (in MMIsar0) MMI_ppncant:
  shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) + ( C - B ) ) = ( A + C )
proof -
  have S1: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( A + B ) = ( B + A ) by (rule MMI_axaddcom)
  from S1 have S2: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + B ) = ( B + A ) by (rule MMI_3adant3)
  from S2 have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( B - C ) ) =
 ( ( B + A ) - ( B - C ) ) by (rule MMI_opreq1d)
  have S4: ( ( A + B ) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A + B ) - ( B - C ) ) =
 ( ( A + B ) + ( C - B ) ) by (rule MMI_subsub2t)
  have S5: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A + B ) ∈ ℂ by (rule MMI_axaddcl)
  from S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A + B ) ∈ ℂ by (rule MMI_3adant3)
  have S7: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶ B ∈ ℂ by (rule MMI_3simp2)
```

have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ C ∈ ℂ **by** (rule MMI_3simp3)
from S4 S6 S7 S8 **have** S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (B - C)) =
((A + B) + (C - B)) **by** (rule MMI_syl3anc)
have S10: (B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) - (B - C)) = (A + C) **by** (rule MMI_pnncant)
from S10 **have** S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((B + A) - (B - C)) = (A + C) **by** (rule MMI_3com12)
from S3 S9 S11 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) + (C - B)) = (A + C) **by** (rule MMI_3eqtr3d)
qed

lemma (**in** MMIsar0) MMI_pnncan: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ **and**
A3: C ∈ ℂ
shows ((A + B) - (A - C)) = (B + C)
proof -
from A1 **have** S1: A ∈ ℂ.
from A2 **have** S2: B ∈ ℂ.
from A3 **have** S3: C ∈ ℂ.
have S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + B) - (A - C)) = (B + C) **by** (rule MMI_pnncant)
from S1 S2 S3 S4 **show** ((A + B) - (A - C)) = (B + C) **by** (rule
MMI_mp3an)
qed

lemma (**in** MMIsar0) MMI_mulcan: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ **and**
A3: C ∈ ℂ **and**
A4: A ≠ **0**
shows (A · B) = (A · C) ⟷ B = C
proof -
from A1 **have** S1: A ∈ ℂ.
from A4 **have** S2: A ≠ **0**.
from S1 S2 **have** S3: ∃ x ∈ ℂ . (A · x) = **1** **by** (rule MMI_recex)
from A1 **have** S4: A ∈ ℂ.
from A2 **have** S5: B ∈ ℂ.
{ **fix** x
have S6: (x ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
((x · A) · B) = (x · (A · B)) **by** (rule MMI_axmulass)
from S5 S6 **have** S7: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
((x · A) · B) = (x · (A · B)) **by** (rule MMI_mp3an3)
from A3 **have** S8: C ∈ ℂ.
have S9: (x ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ⟶
((x · A) · C) = (x · (A · C)) **by** (rule MMI_axmulass)
from S8 S9 **have** S10: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
((x · A) · C) = (x · (A · C)) **by** (rule MMI_mp3an3)
from S7 S10 **have** S11: (x ∈ ℂ ∧ A ∈ ℂ) ⟶
(((x · A) · B) =

((x · A) · C) ⟷
(x · (A · B)) =
(x · (A · C))) **by** (rule MMI_eqeq12d)
from S4 S11 **have** S12: x ∈ ℂ ⟶
(((x · A) · B) =
((x · A) · C) ⟷
(x · (A · B)) =
(x · (A · C))) **by** (rule MMI_mpan2)
have S13:
(A · B) = (A · C) ⟶
(x · (A · B)) = (x · (A · C)) **by** (rule MMI_opreq2)
from S12 S13 **have** S14: x ∈ ℂ ⟶
((A · B) = (A · C) ⟶ ((x · A) · B) =
((x · A) · C)) **by** (rule MMI_syl5bir)
from S14 **have** S15:
(x ∈ ℂ ∧ (A · x) = **1**) ⟶ ((A · B) =
(A · C) ⟶ ((x · A) · B) =
((x · A) · C)) **by** (rule MMI_adantr)
from A1 **have** S16: A ∈ ℂ.
have S17: (A ∈ ℂ ∧ x ∈ ℂ) ⟶
(A · x) = (x · A) **by** (rule MMI_axmulcom)
from S16 S17 **have** S18: x ∈ ℂ ⟶ (A · x) = (x · A)
by (rule MMI_mpan)
from S18 **have** S19: x ∈ ℂ ⟶
((A · x) = **1** ⟷ (x · A) = **1**) **by** (rule MMI_eqeq1d)
have S20: (x · A) =
1 ⟶ ((x · A) · B) = (**1** · B) **by** (rule MMI_opreq1)
from A2 **have** S21: B ∈ ℂ.
from S21 **have** S22: (**1** · B) = B **by** (rule MMI_mulid2)
from S20 S22 **have** S23: (x · A) = **1** ⟶ ((x · A) · B) = B
by (rule MMI_syl6eq)
have S24: (x · A) =
1 ⟶ ((x · A) · C) = (**1** · C) **by** (rule MMI_opreq1)
from A3 **have** S25: C ∈ ℂ.
from S25 **have** S26: (**1** · C) = C **by** (rule MMI_mulid2)
from S24 S26 **have** S27: (x · A) = **1** ⟶ ((x · A) · C) = C
by (rule MMI_syl6eq)
from S23 S27 **have** S28: (x · A) = **1** ⟶
(((x · A) · B) =
((x · A) · C) ⟷ B = C) **by** (rule MMI_eqeq12d)
from S19 S28 **have** S29: x ∈ ℂ ⟶
((A · x) = **1** ⟶
(((x · A) · B) =
((x · A) · C) ⟷ B = C)) **by** (rule MMI_syl6bi)
from S29 **have** S30:
(x ∈ ℂ ∧ (A · x) = **1**) ⟶
(((x · A) · B) =
((x · A) · C) ⟷ B = C) **by** (rule MMI_imp)
from S15 S30 **have** S31:

```
      ( x ∈ ℂ ∧ ( A · x ) = 1 ) ⟶
      ( ( A · B ) = ( A · C ) ⟶ B = C ) by (rule MMI_sylibd)
    from S31 have x ∈ ℂ ⟶
      ( ( A · x ) = 1 ⟶  ( ( A · B ) = ( A · C ) ⟶ B = C ) )
      by (rule MMI_ex)
    } then have  S32: ∀ x. x ∈ ℂ ⟶
      ( ( A · x ) = 1 ⟶  ( ( A · B ) = ( A · C ) ⟶ B = C ) )
      by auto
    from S32 have S33: ( ∃ x ∈ ℂ . ( A · x ) =  1 ) ⟶
      ( ( A · B ) = ( A · C ) ⟶ B = C ) by (rule MMI_r19_23aiv)
    from S3 S33 have S34: ( A · B ) = ( A · C ) ⟶ B = C
      by (rule MMI_ax_mp)
    have S35: B = C ⟶ ( A · B ) = ( A · C ) by (rule MMI_opreq2)
    from S34 S35 show ( A · B ) = ( A · C ) ⟷ B = C by (rule MMI_impbi)
qed

lemma (in MMIsar0) MMI_mulcant2: assumes A1: A ≠ 0
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( ( A · B ) = ( A · C ) ⟷ B = C )
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( A · B ) =
 ( if ( A ∈ ℂ , A , 1 ) · B ) by (rule MMI_opreq1)
   have S2: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( A · C ) =
 ( if ( A ∈ ℂ , A , 1 ) · C ) by (rule MMI_opreq1)
   from S1 S2 have S3: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( ( A · B ) =
 ( A · C ) ⟷
 ( if ( A ∈ ℂ , A , 1 ) · B ) =
 ( if ( A ∈ ℂ , A , 1 ) · C ) ) by (rule MMI_eqeq12d)
   from S3 have S4: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( ( ( A · B ) = ( A · C ) ⟷ B = C ) ⟷
 ( ( if ( A ∈ ℂ , A , 1 ) · B ) =
 ( if ( A ∈ ℂ , A , 1 ) · C ) ⟷
 B = C ) ) by (rule MMI_bibi1d)
   have S5: B =
 if ( B ∈ ℂ , B , 1 ) ⟶
 ( if ( A ∈ ℂ , A , 1 ) · B ) =
 ( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) by (rule MMI_opreq2)
   from S5 have S6: B =
 if ( B ∈ ℂ , B , 1 ) ⟶
 ( ( if ( A ∈ ℂ , A , 1 ) · B ) =
 ( if ( A ∈ ℂ , A , 1 ) · C ) ⟷
 ( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) =
```

```
( if ( A ∈ ℂ , A , 1 ) · C ) ) by (rule MMI_eqeq1d)
  have S7: B =
if ( B ∈ ℂ , B , 1 ) ⟶
( B = C ⟷ if ( B ∈ ℂ , B , 1 ) = C ) by (rule MMI_eqeq1)
  from S6 S7 have S8: B =
if ( B ∈ ℂ , B , 1 ) ⟶
( ( ( if ( A ∈ ℂ , A , 1 ) · B ) = ( if ( A ∈ ℂ , A , 1 ) · C ) ⟷
B = C ) ⟷
( ( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) =
( if ( A ∈ ℂ , A , 1 ) · C ) ⟷
if ( B ∈ ℂ , B , 1 ) = C ) ) by (rule MMI_bibi12d)
  have S9: C =
if ( C ∈ ℂ , C , 1 ) ⟶
( if ( A ∈ ℂ , A , 1 ) · C ) =
( if ( A ∈ ℂ , A , 1 ) · if ( C ∈ ℂ , C , 1 ) ) by (rule MMI_opreq2)
  from S9 have S10: C =
if ( C ∈ ℂ , C , 1 ) ⟶
( ( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) =
( if ( A ∈ ℂ , A , 1 ) · C ) ⟷
( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) =
( if ( A ∈ ℂ , A , 1 ) · if ( C ∈ ℂ , C , 1 ) ) ) by (rule MMI_eqeq2d)
  have S11: C =
if ( C ∈ ℂ , C , 1 ) ⟶
( if ( B ∈ ℂ , B , 1 ) =
C ⟷
if ( B ∈ ℂ , B , 1 ) =
if ( C ∈ ℂ , C , 1 ) ) by (rule MMI_eqeq2)
  from S10 S11 have S12: C =
if ( C ∈ ℂ , C , 1 ) ⟶
( ( ( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) = ( if ( A ∈ ℂ ,
A , 1 ) · C ) ⟷ if ( B ∈ ℂ , B , 1 ) = C ) ⟷
( ( if ( A ∈ ℂ , A , 1 ) · if ( B ∈ ℂ , B , 1 ) ) =
( if ( A ∈ ℂ , A , 1 ) · if ( C ∈ ℂ , C , 1 ) ) ⟷
if ( B ∈ ℂ , B , 1 ) =
if ( C ∈ ℂ , C , 1 ) ) ) by (rule MMI_bibi12d)
  have S13: 1 ∈ ℂ by (rule MMI_1cn)
  from S13 have S14: if ( A ∈ ℂ , A , 1 ) ∈ ℂ by (rule MMI_elimel)
  have S15: 1 ∈ ℂ by (rule MMI_1cn)
  from S15 have S16: if ( B ∈ ℂ , B , 1 ) ∈ ℂ by (rule MMI_elimel)
  have S17: 1 ∈ ℂ by (rule MMI_1cn)
  from S17 have S18: if ( C ∈ ℂ , C , 1 ) ∈ ℂ by (rule MMI_elimel)
  have S19: A =
if ( A ∈ ℂ , A , 1 ) ⟶
( A ≠ 0 ⟷ if ( A ∈ ℂ , A , 1 ) ≠ 0 ) by (rule MMI_neeq1)
  have S20: 1 =
if ( A ∈ ℂ , A , 1 ) ⟶
( 1 ≠ 0 ⟷ if ( A ∈ ℂ , A , 1 ) ≠ 0 ) by (rule MMI_neeq1)
  from A1 have S21: A ≠ 0.
  have S22: 1 ≠ 0 by (rule MMI_ax1ne0)
```

from S19 S20 S21 S22 **have** S23: if (A ∈ ℂ , A , **1**) ≠ **0** **by** (rule MMI_keephyp)
from S14 S16 S18 S23 **have** S24: (if (A ∈ ℂ , A , **1**) · if (B ∈ ℂ , B , **1**)) =
(if (A ∈ ℂ , A , **1**) · if (C ∈ ℂ , C , **1**)) ⟷
if (B ∈ ℂ , B , **1**) =
if (C ∈ ℂ , C , **1**) **by** (rule MMI_mulcan)
from S4 S8 S12 S24 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) = (A · C) ⟷ B = C) **by** (rule MMI_dedth3h)
qed

lemma (**in** MMIsar0) MMI_mulcant:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ A ≠ **0**) ⟶
((A · B) = (A · C) ⟷ B = C)
proof -
have S1: A =
if (A ≠ **0** , A , **1**) ⟶
(A ∈ ℂ ⟷ if (A ≠ **0** , A , **1**) ∈ ℂ) **by** (rule MMI_eleq1)
have S2: A =
if (A ≠ **0** , A , **1**) ⟶
(B ∈ ℂ ⟷ B ∈ ℂ) **by** (rule MMI_pm4_2i)
have S3: A =
if (A ≠ **0** , A , **1**) ⟶
(C ∈ ℂ ⟷ C ∈ ℂ) **by** (rule MMI_pm4_2i)
from S1 S2 S3 **have** S4: A =
if (A ≠ **0** , A , **1**) ⟶
((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟷
(if (A ≠ **0** , A , **1**) ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ)) **by** (rule MMI_3anbi123d)
have S5: A =
if (A ≠ **0** , A , **1**) ⟶
(A · B) =
(if (A ≠ **0** , A , **1**) · B) **by** (rule MMI_opreq1)
have S6: A =
if (A ≠ **0** , A , **1**) ⟶
(A · C) =
(if (A ≠ **0** , A , **1**) · C) **by** (rule MMI_opreq1)
from S5 S6 **have** S7: A =
if (A ≠ **0** , A , **1**) ⟶
((A · B) =
(A · C) ⟷
(if (A ≠ **0** , A , **1**) · B) =
(if (A ≠ **0** , A , **1**) · C)) **by** (rule MMI_eqeq12d)
from S7 **have** S8: A =
if (A ≠ **0** , A , **1**) ⟶
(((A · B) = (A · C) ⟷ B = C) ⟷
((if (A ≠ **0** , A , **1**) · B) =
(if (A ≠ **0** , A , **1**) · C) ⟷
B = C)) **by** (rule MMI_bibi1d)
from S4 S8 **have** S9: A =

if (A $\neq$ **0** , A , **1**) $\longrightarrow$
(((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$ ((A $\cdot$ B) = (A $\cdot$ C) $\longleftrightarrow$ B = C)) $\longleftrightarrow$
((if (A $\neq$ **0** , A , **1**) $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((if (A $\neq$ **0** , A , **1**) $\cdot$ B) =
(if (A $\neq$ **0** , A , **1**) $\cdot$ C) $\longleftrightarrow$
B = C))) **by** (rule MMI_imbi12d)
have S10: if (A $\neq$ **0** , A , **1**) $\neq$ **0** **by** (rule MMI_elimne0)
from S10 **have** S11: (if (A $\neq$ **0** , A , **1**) $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((if (A $\neq$ **0** , A , **1**) $\cdot$ B) =
(if (A $\neq$ **0** , A , **1**) $\cdot$ C) $\longleftrightarrow$ B = C) **by** (rule MMI_mulcant2)
from S9 S11 **have** S12: A $\neq$ **0** $\longrightarrow$
((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A $\cdot$ B) = (A $\cdot$ C) $\longleftrightarrow$ B = C)) **by** (rule MMI_dedth)
from S12 **show** ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ A $\neq$ **0**) $\longrightarrow$
((A $\cdot$ B) = (A $\cdot$ C) $\longleftrightarrow$ B = C) **by** (rule MMI_impcom)
qed

lemma (**in** MMIsar0) MMI_mulcan2t:
shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ C) = (B $\cdot$ C) $\longleftrightarrow$ A = B)
proof -
have S1: (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A $\cdot$ C) = (C $\cdot$ A) **by** (rule MMI_axmulcom)
from S1 **have** S2: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A $\cdot$ C) = (C $\cdot$ A) **by** (rule MMI_3adant2)
have S3: (B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(B $\cdot$ C) = (C $\cdot$ B) **by** (rule MMI_axmulcom)
from S3 **have** S4: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(B $\cdot$ C) = (C $\cdot$ B) **by** (rule MMI_3adant1)
from S2 S4 **have** S5: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A $\cdot$ C) =
(B $\cdot$ C) $\longleftrightarrow$ (C $\cdot$ A) = (C $\cdot$ B)) **by** (rule MMI_eqeq12d)
from S5 **have** S6: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ C) =
(B $\cdot$ C) $\longleftrightarrow$ (C $\cdot$ A) = (C $\cdot$ B)) **by** (rule MMI_adantr)
have S7: ((C $\in$ $\mathbb{C}$ $\wedge$ A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B) **by** (rule MMI_mulcant)
from S7 **have** S8: (C $\in$ $\mathbb{C}$ $\wedge$ A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
(C $\neq$ **0** $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B)) **by** (rule MMI_ex)
from S8 **have** S9: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(C $\neq$ **0** $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B)) **by** (rule MMI_3coml)
from S9 **have** S10: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((C $\cdot$ A) = (C $\cdot$ B) $\longleftrightarrow$ A = B) **by** (rule MMI_imp)
from S6 S10 **show** ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ C) = (B $\cdot$ C) $\longleftrightarrow$ A = B) **by** (rule MMI_bitrd)

qed

lemma (**in** MMIsar0) MMI_mul0or: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (A · B) = **0** ⟷ (A = **0** ∨ B = **0**)
proof -
have S1: A ≠ **0** ⟷ ¬ (A = **0**) **by** (rule MMI_df_ne)
from A1 **have** S2: A ∈ ℂ.
from A2 **have** S3: B ∈ ℂ.
have S4: **0** ∈ ℂ **by** (rule MMI_0cn)
from S2 S3 S4 **have** S5: A ∈ ℂ ∧ B ∈ ℂ ∧ **0** ∈ ℂ **by** (rule MMI_3pm3_2i)
have S6: ((A ∈ ℂ ∧ B ∈ ℂ ∧ **0** ∈ ℂ) ∧ A ≠ **0**) ⟶
((A · B) = (A · **0**) ⟷ B = **0**) **by** (rule MMI_mulcant)
from S5 S6 **have** S7: A ≠ **0** ⟶
((A · B) = (A · **0**) ⟷ B = **0**) **by** (rule MMI_mpan)
from A1 **have** S8: A ∈ ℂ.
from S8 **have** S9: (A · **0**) = **0** **by** (rule MMI_mul01)
from S9 **have** S10: (A · B) = (A · **0**) ⟷ (A · B) = **0** **by** (rule
MMI_eqeq2i)
from S7 S10 **have** S11: A ≠ **0** ⟶ ((A · B) = **0** ⟷ B = **0**) **by** (rule
MMI_syl5bbr)
from S11 **have** S12: A ≠ **0** ⟶ ((A · B) = **0** ⟶ B = **0**) **by** (rule
MMI_biimpd)
from S1 S12 **have** S13: ¬ (A =
0) ⟶ ((A · B) = **0** ⟶ B = **0**) **by** (rule MMI_sylbir)
from S13 **have** S14: (A · B) =
0 ⟶ (¬ (A = **0**) ⟶ B = **0**) **by** (rule MMI_com12)
from S14 **have** S15: (A · B) = **0** ⟶ (A = **0** ∨ B = **0**) **by** (rule MMI_orrd)
have S16: A = **0** ⟶ (A · B) = (**0** · B) **by** (rule MMI_opreq1)
from A2 **have** S17: B ∈ ℂ.
from S17 **have** S18: (**0** · B) = **0** **by** (rule MMI_mul02)
from S16 S18 **have** S19: A = **0** ⟶ (A · B) = **0** **by** (rule MMI_syl6eq)
have S20: B = **0** ⟶ (A · B) = (A · **0**) **by** (rule MMI_opreq2)
from S9 **have** S21: (A · **0**) = **0** .
from S20 S21 **have** S22: B = **0** ⟶ (A · B) = **0** **by** (rule MMI_syl6eq)
from S19 S22 **have** S23: (A = **0** ∨ B = **0**) ⟶ (A · B) = **0** **by** (rule
MMI_jaoi)
from S15 S23 **show** (A · B) = **0** ⟷ (A = **0** ∨ B = **0**) **by** (rule MMI_impbi)
qed

lemma (**in** MMIsar0) MMI_msq0: **assumes** A1: A ∈ ℂ
shows (A · A) = **0** ⟷ A = **0**
proof -
from A1 **have** S1: A ∈ ℂ.
from A1 **have** S2: A ∈ ℂ.
from S1 S2 **have** S3: (A · A) = **0** ⟷ (A = **0** ∨ A = **0**) **by** (rule
MMI_mul0or)
have S4: (A = **0** ∨ A = **0**) ⟷ A = **0** **by** (rule MMI_oridm)
from S3 S4 **show** (A · A) = **0** ⟷ A = **0** **by** (rule MMI_bitr)

qed

```
lemma (in MMIsar0) MMI_mul0ort:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( A · B ) = 0 ⟷ ( A = 0 ∨ B = 0 ) )
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A · B ) =
 ( if ( A ∈ ℂ , A , 0 ) · B ) by (rule MMI_opreq1)
   from S1 have S2: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( A · B ) =
 0 ⟷ ( if ( A ∈ ℂ , A , 0 ) · B ) = 0 ) by (rule MMI_eqeq1d)
   have S3: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A = 0 ⟷ if ( A ∈ ℂ , A , 0 ) = 0 ) by (rule MMI_eqeq1)
   from S3 have S4: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( A = 0 ∨ B = 0 ) ⟷
 ( if ( A ∈ ℂ , A , 0 ) = 0 ∨ B = 0 ) ) by (rule MMI_orbi1d)
   from S2 S4 have S5: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( ( A · B ) = 0 ⟷ ( A = 0 ∨ B = 0 ) ) ⟷
 ( ( if ( A ∈ ℂ , A , 0 ) · B ) =
 0 ⟷
 ( if ( A ∈ ℂ , A , 0 ) =
 0 ∨ B = 0 ) ) ) by (rule MMI_bibi12d)
   have S6: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( if ( A ∈ ℂ , A , 0 ) · B ) =
 ( if ( A ∈ ℂ , A , 0 ) · if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
   from S6 have S7: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( if ( A ∈ ℂ , A , 0 ) · B ) =
 0 ⟷
 ( if ( A ∈ ℂ , A , 0 ) · if ( B ∈ ℂ , B , 0 ) ) =
 0 ) by (rule MMI_eqeq1d)
   have S8: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( B = 0 ⟷ if ( B ∈ ℂ , B , 0 ) = 0 ) by (rule MMI_eqeq1)
   from S8 have S9: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( if ( A ∈ ℂ , A , 0 ) = 0 ∨ B = 0 ) ⟷
 ( if ( A ∈ ℂ , A , 0 ) =
 0 ∨ if ( B ∈ ℂ , B , 0 ) = 0 ) ) by (rule MMI_orbi2d)
   from S7 S9 have S10: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( ( if ( A ∈ ℂ , A , 0 ) · B ) = 0 ⟷ ( if ( A ∈ ℂ , A , 0 ) = 0
```

∨ B = 0)) ⟷
((if (A ∈ ℂ , A , 0) · if (B ∈ ℂ , B , 0)) =
0 ⟷
(if (A ∈ ℂ , A , 0) =
0 ∨ if (B ∈ ℂ , B , 0) = 0))) **by** (rule MMI_bibi12d)
have S11: 0 ∈ ℂ **by** (rule MMI_0cn)
from S11 **have** S12: if (A ∈ ℂ , A , 0) ∈ ℂ **by** (rule MMI_elimel)
have S13: 0 ∈ ℂ **by** (rule MMI_0cn)
from S13 **have** S14: if (B ∈ ℂ , B , 0) ∈ ℂ **by** (rule MMI_elimel)
from S12 S14 **have** S15: (if (A ∈ ℂ , A , 0) · if (B ∈ ℂ , B , 0
)) =
0 ⟷
(if (A ∈ ℂ , A , 0) =
0 ∨ if (B ∈ ℂ , B , 0) = 0) **by** (rule MMI_mul0or)
from S5 S10 S15 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A · B) = 0 ⟷ (A = 0 ∨ B = 0)) **by** (rule MMI_dedth2h)
qed

lemma (**in** MMIsar0) MMI_muln0bt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A ≠ 0 ∧ B ≠ 0) ⟷ (A · B) ≠ 0)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A · B) = 0 ⟷ (A = 0 ∨ B = 0)) **by** (rule MMI_mul0ort)
from S1 **have** S2: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(¬ ((A · B) = 0) ⟷
¬ ((A = 0 ∨ B = 0))) **by** (rule MMI_negbid)
have S3: ¬ ((A = 0 ∨ B = 0)) ⟷
(¬ (A = 0) ∧ ¬ (B = 0)) **by** (rule MMI_ioran)
from S2 S3 **have** S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((¬ (A = 0) ∧ ¬ (B = 0)) ⟷
¬ ((A · B) = 0)) **by** (rule MMI_syl6rbb)
have S5: A ≠ 0 ⟷ ¬ (A = 0) **by** (rule MMI_df_ne)
have S6: B ≠ 0 ⟷ ¬ (B = 0) **by** (rule MMI_df_ne)
from S5 S6 **have** S7: (A ≠ 0 ∧ B ≠ 0) ⟷
(¬ (A = 0) ∧ ¬ (B = 0)) **by** (rule MMI_anbi12i)
have S8: (A · B) ≠ 0 ⟷ ¬ ((A · B) = 0) **by** (rule MMI_df_ne)
from S4 S7 S8 **show** (A ∈ ℂ ∧ B ∈ ℂ) ⟶
((A ≠ 0 ∧ B ≠ 0) ⟷ (A · B) ≠ 0) **by** (rule MMI_3bitr4g)
qed

lemma (**in** MMIsar0) MMI_muln0: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ **and**
A3: A ≠ 0 **and**
A4: B ≠ 0
shows (A · B) ≠ 0
proof -

```
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: A ≠ 0.
   from A4 have S4: B ≠ 0.
   from S3 S4 have S5: A ≠ 0 ∧ B ≠ 0 by (rule MMI_pm3_2i)
   have S6: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( ( A ≠ 0 ∧ B ≠ 0 ) ⟷ ( A · B ) ≠ 0 ) by (rule MMI_muln0bt)
   from S5 S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( A · B ) ≠ 0 by (rule
MMI_mpbii)
   from S1 S2 S7 show ( A · B ) ≠ 0 by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_receu: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: A ≠ 0
   shows ∃! x . x ∈ ℂ ∧ ( A · x ) = B
proof -
  { fix x y
    have S1: x = y ⟶ ( A · x ) = ( A · y ) by (rule MMI_opreq2)
    from S1 have S2: x = y ⟶ ( ( A · x ) = B ⟷ ( A · y ) = B )
      by (rule MMI_eqeq1d)
  } then have S2: ∀ x y. x = y ⟶ ( ( A · x ) = B ⟷ ( A · y ) = B
)
    by simp
    from S2 have S3:
      ( ∃! x . x ∈ ℂ ∧ ( A · x ) =  B ) ⟷
      ( ( ∃ x ∈ ℂ . ( A · x ) = B ) ∧
      ( ∀ x ∈ ℂ . ∀ y ∈ ℂ . ( ( ( A · x ) = B ∧ ( A · y ) = B ) ⟶ x
= y ) ) )
      by (rule MMI_reu4)
    from A1 have S4: A ∈ ℂ.
    from A3 have S5: A ≠ 0.
   from S4 S5 have S6: ∃ y ∈ ℂ . ( A · y ) = 1 by (rule MMI_recex)
   from A2 have S7: B ∈ ℂ.
   { fix y
     have S8: ( y ∈ ℂ ∧ B ∈ ℂ ) ⟶ ( y · B ) ∈ ℂ by (rule MMI_axmulcl)
     from S7 S8 have S9: y ∈ ℂ ⟶ ( y · B ) ∈ ℂ by (rule MMI_mpan2)
     have S10: ( y · B ) ∈ ℂ ⟷
       ( ∃ x ∈ ℂ . x = ( y · B ) ) by (rule MMI_risset)
     from S9 S10 have S11: y ∈ ℂ ⟶ ( ∃ x ∈ ℂ . x = ( y · B ) )
       by (rule MMI_sylib)
     { fix x
       have S12: x =  ( y · B ) ⟶
  ( A · x ) = ( A · ( y · B ) ) by (rule MMI_opreq2)
       from A1 have S13: A ∈ ℂ.
       from A2 have S14: B ∈ ℂ.
       have S15: ( A ∈ ℂ ∧ y ∈ ℂ ∧ B ∈ ℂ ) ⟶
  ( ( A · y ) · B ) = ( A · ( y · B ) ) by (rule MMI_axmulass)
       from S13 S14 S15 have S16: y ∈ ℂ ⟶
```

((A · y) · B) = (A · (y · B)) **by** (rule MMI_mp3an13)
    from S16 **have** S17: y ∈ ℂ ⟶
(A · (y · B)) = ((A · y) · B) **by** (rule MMI_eqcomd)
    from S12 S17 **have** S18: (y ∈ ℂ ∧ x =
(y · B)) ⟶
(A · x) = ((A · y) · B) **by** (rule MMI_sylan9eqr)
    have S19: (A · y) =
1 ⟶ ((A · y) · B) = (**1** · B) **by** (rule MMI_opreq1)
    from A2 **have** S20: B ∈ ℂ.
    from S20 **have** S21: (**1** · B) = B **by** (rule MMI_mulid2)
    from S19 S21 **have** S22: (A · y) = **1** ⟶ ((A · y) · B) = B
by (rule MMI_syl6eq)
    from S18 S22 **have** S23:
((A · y) = **1** ∧ (y ∈ ℂ ∧ x =
(y · B))) ⟶ (A · x) = B **by** (rule MMI_sylan9eqr)
    from S23 **have** S24:
(A · y) = **1** ⟶ (y ∈ ℂ ⟶
(x = (y · B) ⟶ (A · x) = B)) **by** (rule MMI_exp32)
    from S24 **have** S25: (y ∈ ℂ ∧ (A · y) =
1) ⟶
(x = (y · B) ⟶ (A · x) = B) **by** (rule MMI_impcom)
    from S25 **have**
(y ∈ ℂ ∧ (A · y) = **1**) ⟶ (x ∈ ℂ ⟶
(x = (y · B) ⟶ (A · x) = B)) **by** (rule MMI_a1d)
    } **then have** S26:
∀ x . (y ∈ ℂ ∧ (A · y) = **1**) ⟶ (x ∈ ℂ ⟶
(x = (y · B) ⟶ (A · x) = B)) **by** simp
    from S26 **have** S27:
(y ∈ ℂ ∧ (A · y) = **1**) ⟶
(∀ x ∈ ℂ . (x = (y · B) ⟶ (A · x) = B)) **by** (rule MMI_r19_21aiv)
    from S27 **have** S28: y ∈ ℂ ⟶
((A · y) = **1** ⟶
(∀ x ∈ ℂ . (x = (y · B) ⟶ (A · x) = B))) **by** (rule MMI_ex)
    have S29: (∀ x ∈ ℂ . (x = (y · B) ⟶ (A · x) = B)) ⟶

((∃ x ∈ ℂ . x = (y · B)) ⟶
(∃ x ∈ ℂ . (A · x) = B)) **by** (rule MMI_r19_22)
    from S28 S29 **have** S30:
y ∈ ℂ ⟶ ((A · y) = **1** ⟶
((∃ x ∈ ℂ . x = (y · B)) ⟶
(∃ x ∈ ℂ . (A · x) = B))) **by** (rule MMI_syl6)
    from S11 S30 **have**
y ∈ ℂ ⟶ ((A · y) = **1** ⟶ (∃ x ∈ ℂ . (A · x) = B))
by (rule MMI_mpid)
    } **then have** S31:
  ∀ y . y ∈ ℂ ⟶ ((A · y) = **1** ⟶ (∃ x ∈ ℂ . (A · x) = B)
)
by simp
    from S31 **have** S32: (∃ y ∈ ℂ . (A · y) =

```
  1 ) ⟶ ( ∃ x ∈ ℂ . ( A · x ) = B ) by (rule MMI_r19_23aiv)
       from S6 S32 have S33: ∃ x ∈ ℂ . ( A · x ) = B by (rule MMI_ax_mp)
       from A1 have S34: A ∈ ℂ.
       from A3 have S35: A ≠ 0.
       { fix x y
  from S35 have S36: ( A ∈ ℂ ∧ x ∈ ℂ ∧ y ∈ ℂ ) ⟶
    ( ( A · x ) = ( A · y ) ⟷ x = y ) by (rule MMI_mulcant2)
  have S37:
    ( ( A · x ) =  B ∧ ( A · y ) =
    B ) ⟶ ( A · x ) = ( A · y ) by (rule MMI_eqtr3t)
  from S36 S37 have S38: ( A ∈ ℂ ∧ x ∈ ℂ ∧ y ∈ ℂ ) ⟶
    ( ( ( A · x ) = B ∧ ( A · y ) = B ) ⟶
    x = y ) by (rule MMI_syl5bi)
  from S34 S38 have ( x ∈ ℂ ∧ y ∈ ℂ ) ⟶
    ( ( ( A · x ) = B ∧ ( A · y ) = B ) ⟶
    x = y ) by (rule MMI_mp3an1)
       } then have  S39: ∀ x y. ( x ∈ ℂ ∧ y ∈ ℂ ) ⟶
    ( ( ( A · x ) = B ∧ ( A · y ) = B ) ⟶
    x = y ) by auto
       from S39 have S40:
  ∀ x ∈ ℂ . ∀ y ∈ ℂ . ( ( ( A · x ) = B ∧ ( A · y ) = B ) ⟶
  x = y ) by (rule MMI_rgen2)
       from S3 S33 S40 show ∃! x . x ∈ ℂ ∧ ( A · x ) = B by (rule MMI_mpbir2an)
qed
```

```
lemma (in MMIsar0) MMI_divval: assumes A ∈ ℂ  B ∈ ℂ B ≠ 0
  shows A / B = ⋃ { x ∈ ℂ . B · x = A }
  using cdiv_def by simp
```

```
lemma (in MMIsar0) MMI_divmul: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ and
    A4: B ≠ 0
   shows ( A / B ) = C ⟷ ( B · C ) = A
proof -
   from A3 have S1: C ∈ ℂ.
   { fix x
     have S2: x =
       C ⟶ ( ( A / B ) = x ⟷ ( A / B ) = C ) by (rule MMI_eqeq2)
     have S3: x = C ⟶ ( B · x ) = ( B · C ) by (rule MMI_opreq2)
     from S3 have S4: x =
       C ⟶ ( ( B · x ) = A ⟷ ( B · C ) = A ) by (rule MMI_eqeq1d)
     from S2 S4 have
       x = C ⟶
```

```
        ( ( ( A / B ) = x ⟷ ( B · x ) = A ) ⟷
        ( ( A / B ) = C ⟷ ( B · C ) = A ) ) by (rule MMI_bibi12d)
   } then have S5: ∀x. x = C ⟶
        ( ( ( A / B ) = x ⟷ ( B · x ) = A ) ⟷
        ( ( A / B ) = C ⟷ ( B · C ) = A ) )
     by simp
   from A2 have S6: B ∈ ℂ.
   from A1 have S7: A ∈ ℂ.
   from A4 have S8: B ≠ 0.
   from S6 S7 S8 have S9: ∃! x . x ∈ ℂ ∧ ( B · x ) = A by (rule MMI_receu)
   { fix x
     have S10: ( x ∈ ℂ ∧ ( ∃! x . x ∈ ℂ ∧ ( B · x ) = A ) ) ⟶
        ( ( B · x ) =
        A ⟷ ⋃ { x ∈ ℂ . ( B · x ) = A } = x ) by (rule MMI_reuuni1)
     from S9 S10 have
        x ∈ ℂ ⟶ ( ( B · x ) =  A ⟷ ⋃ { x ∈ ℂ . ( B · x ) = A } =
x )
        by (rule MMI_mpan2)
   } then have S11:
        ∀ x. x ∈ ℂ ⟶ ( ( B · x ) =  A ⟷ ⋃ { x ∈ ℂ . ( B · x ) = A
} = x )
     by blast
   from A1 have S12: A ∈ ℂ.
   from A2 have S13: B ∈ ℂ.
   from A4 have S14: B ≠ 0.
   from S12 S13 S14 have S15: ( A / B ) =
     ⋃ { x ∈ ℂ . ( B · x ) = A } by (rule MMI_divval)
   from S15 have S16: ∀ x. ( A / B ) =
     x ⟷ ⋃ { x ∈ ℂ . ( B · x ) = A } = x by simp
   from S11 S16 have S17: ∀ x. x ∈ ℂ ⟶
     ( ( A / B ) = x ⟷ ( B · x ) = A ) by (rule MMI_syl6rbbr)
   from S5 S17 have S18: C ∈ ℂ ⟶
     ( ( A / B ) = C ⟷ ( B · C ) = A ) by (rule MMI_vtoclga)
   from S1 S18 show ( A / B ) = C ⟷ ( B · C ) = A by (rule MMI_ax_mp)
qed

lemma (in MMIsar0) MMI_divmulz: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
   shows B ≠ 0 ⟶
 ( ( A / B ) = C ⟷ ( B · C ) = A )
proof -
   have S1: B =
 if ( B ≠ 0 , B , 1 ) ⟶
 ( A / B ) =
 ( A / if ( B ≠ 0 , B , 1 ) ) by (rule MMI_opreq2)
   from S1 have S2: B =
 if ( B ≠ 0 , B , 1 ) ⟶
 ( ( A / B ) =
```

C $\longleftrightarrow$ (A / if (B $\neq$ **0** , B , **1**)) = C) **by** (rule MMI_eqeq1d)
  have S3: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(B $\cdot$ C) =
(if (B $\neq$ **0** , B , **1**) $\cdot$ C) **by** (rule MMI_opreq1)
  from S3 **have** S4: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
((B $\cdot$ C) =
A $\longleftrightarrow$ (if (B $\neq$ **0** , B , **1**) $\cdot$ C) = A) **by** (rule MMI_eqeq1d)
  from S2 S4 **have** S5: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) $\longleftrightarrow$
((A / if (B $\neq$ **0** , B , **1**)) =
C $\longleftrightarrow$
(if (B $\neq$ **0** , B , **1**) $\cdot$ C) = A)) **by** (rule MMI_bibi12d)
  from A1 **have** S6: A $\in$ $\mathbb{C}$.
  from A2 **have** S7: B $\in$ $\mathbb{C}$.
  have S8: **1** $\in$ $\mathbb{C}$ **by** (rule MMI_1cn)
  from S7 S8 **have** S9: if (B $\neq$ **0** , B , **1**) $\in$ $\mathbb{C}$ **by** (rule MMI_keepel)
  from A3 **have** S10: C $\in$ $\mathbb{C}$.
  have S11: if (B $\neq$ **0** , B , **1**) $\neq$ **0** **by** (rule MMI_elimne0)
  from S6 S9 S10 S11 **have** S12: (A / if (B $\neq$ **0** , B , **1**)) =
C $\longleftrightarrow$ (if (B $\neq$ **0** , B , **1**) $\cdot$ C) = A **by** (rule MMI_divmul)
  from S5 S12 **show** B $\neq$ **0** $\longrightarrow$
((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) **by** (rule MMI_dedth)
qed

lemma (**in** MMIsar0) MMI_divmult:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ B $\neq$ **0**) $\longrightarrow$
((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A)
proof -
  have S1: A =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longrightarrow$
(A / B) =
(if (A $\in$ $\mathbb{C}$, A , **0**) / B) **by** (rule MMI_opreq1)
  from S1 **have** S2: A =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longrightarrow$
((A / B) =
C $\longleftrightarrow$ (if (A $\in$ $\mathbb{C}$, A , **0**) / B) = C) **by** (rule MMI_eqeq1d)
  have S3: A =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longrightarrow$
((B $\cdot$ C) =
A $\longleftrightarrow$ (B $\cdot$ C) = if (A $\in$ $\mathbb{C}$, A , **0**)) **by** (rule MMI_eqeq2)
  from S2 S3 **have** S4: A =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longrightarrow$
(((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A) $\longleftrightarrow$
((if (A $\in$ $\mathbb{C}$, A , **0**) / B) =
C $\longleftrightarrow$
(B $\cdot$ C) = if (A $\in$ $\mathbb{C}$, A , **0**))) **by** (rule MMI_bibi12d)

from S4 **have** S5: A =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longrightarrow$
((B $\neq$ **0** $\longrightarrow$ ((A / B) = C $\longleftrightarrow$ (B $\cdot$ C) = A)) $\longleftrightarrow$
(B $\neq$ **0** $\longrightarrow$
((if (A $\in$ $\mathbb{C}$, A , **0**) / B) =
C $\longleftrightarrow$
(B $\cdot$ C) = if (A $\in$ $\mathbb{C}$, A , **0**)))) **by** (rule MMI_imbi2d)
have S6: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
(B $\neq$ **0** $\longleftrightarrow$ if (B $\in$ $\mathbb{C}$, B , **0**) $\neq$ **0**) **by** (rule MMI_neeq1)
have S7: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
(if (A $\in$ $\mathbb{C}$, A , **0**) / B) =
(if (A $\in$ $\mathbb{C}$, A , **0**) / if (B $\in$ $\mathbb{C}$, B , **0**)) **by** (rule MMI_opreq2)
from S7 **have** S8: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
((if (A $\in$ $\mathbb{C}$, A , **0**) / B) =
C $\longleftrightarrow$
(if (A $\in$ $\mathbb{C}$, A , **0**) / if (B $\in$ $\mathbb{C}$, B , **0**)) =
C) **by** (rule MMI_eqeq1d)
have S9: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
(B $\cdot$ C) =
(if (B $\in$ $\mathbb{C}$, B , **0**) $\cdot$ C) **by** (rule MMI_opreq1)
from S9 **have** S10: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
((B $\cdot$ C) =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longleftrightarrow$
(if (B $\in$ $\mathbb{C}$, B , **0**) $\cdot$ C) =
if (A $\in$ $\mathbb{C}$, A , **0**)) **by** (rule MMI_eqeq1d)
from S8 S10 **have** S11: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
(((if (A $\in$ $\mathbb{C}$, A , **0**) / B) = C $\longleftrightarrow$ (B $\cdot$ C) = if (A $\in$ $\mathbb{C}$, A
, **0**)) $\longleftrightarrow$
((if (A $\in$ $\mathbb{C}$, A , **0**) / if (B $\in$ $\mathbb{C}$, B , **0**)) =
C $\longleftrightarrow$
(if (B $\in$ $\mathbb{C}$, B , **0**) $\cdot$ C) =
if (A $\in$ $\mathbb{C}$, A , **0**))) **by** (rule MMI_bibi12d)
from S6 S11 **have** S12: B =
if (B $\in$ $\mathbb{C}$, B , **0**) $\longrightarrow$
((B $\neq$ **0** $\longrightarrow$ ((if (A $\in$ $\mathbb{C}$, A , **0**) / B) = C $\longleftrightarrow$ (B $\cdot$ C) = if
(A $\in$ $\mathbb{C}$, A , **0**))) $\longleftrightarrow$
(if (B $\in$ $\mathbb{C}$, B , **0**) $\neq$ **0** $\longrightarrow$
((if (A $\in$ $\mathbb{C}$, A , **0**) / if (B $\in$ $\mathbb{C}$, B , **0**)) =
C $\longleftrightarrow$
(if (B $\in$ $\mathbb{C}$, B , **0**) $\cdot$ C) =
if (A $\in$ $\mathbb{C}$, A , **0**)))) **by** (rule MMI_imbi12d)
have S13: C =
if (C $\in$ $\mathbb{C}$, C , **0**) $\longrightarrow$

```
 ( ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) =
 C ⟷
 ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) =
 if ( C ∈ ℂ , C , 0 ) ) by (rule MMI_eqeq2)
   have S14: C =
 if ( C ∈ ℂ , C , 0 ) ⟶
 ( if ( B ∈ ℂ , B , 0 ) · C ) =
 ( if ( B ∈ ℂ , B , 0 ) · if ( C ∈ ℂ , C , 0 ) ) by (rule MMI_opreq2)
   from S14 have S15: C =
 if ( C ∈ ℂ , C , 0 ) ⟶
 ( ( if ( B ∈ ℂ , B , 0 ) · C ) =
 if ( A ∈ ℂ , A , 0 ) ⟷
 ( if ( B ∈ ℂ , B , 0 ) · if ( C ∈ ℂ , C , 0 ) ) =
 if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_eqeq1d)
   from S13 S15 have S16: C =
 if ( C ∈ ℂ , C , 0 ) ⟶
 ( ( ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) = C ⟷ ( if (
B ∈ ℂ , B , 0 ) · C ) = if ( A ∈ ℂ , A , 0 ) ) ⟷
 ( ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) =
 if ( C ∈ ℂ , C , 0 ) ⟷
 ( if ( B ∈ ℂ , B , 0 ) · if ( C ∈ ℂ , C , 0 ) ) =
 if ( A ∈ ℂ , A , 0 ) ) ) by (rule MMI_bibi12d)
   from S16 have S17: C =
 if ( C ∈ ℂ , C , 0 ) ⟶
 ( ( if ( B ∈ ℂ , B , 0 ) ≠ 0 ⟶ ( ( if ( A ∈ ℂ , A , 0 ) / if ( B
∈ ℂ , B , 0 ) ) = C ⟷ ( if ( B ∈ ℂ , B , 0 ) · C ) = if ( A ∈ ℂ ,
A , 0 ) ) ) ⟷
 ( if ( B ∈ ℂ , B , 0 ) ≠ 0 ⟶
 ( ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) =
 if ( C ∈ ℂ , C , 0 ) ⟷
 ( if ( B ∈ ℂ , B , 0 ) · if ( C ∈ ℂ , C , 0 ) ) =
 if ( A ∈ ℂ , A , 0 ) ) ) ) by (rule MMI_imbi2d)
   have S18: 0 ∈ ℂ by (rule MMI_0cn)
   from S18 have S19: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   have S20: 0 ∈ ℂ by (rule MMI_0cn)
   from S20 have S21: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
   have S22: 0 ∈ ℂ by (rule MMI_0cn)
   from S22 have S23: if ( C ∈ ℂ , C , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S19 S21 S23 have S24: if ( B ∈ ℂ , B , 0 ) ≠ 0 ⟶
 ( ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) =
 if ( C ∈ ℂ , C , 0 ) ⟷
 ( if ( B ∈ ℂ , B , 0 ) · if ( C ∈ ℂ , C , 0 ) ) =
 if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_divmulz)
   from S5 S12 S17 S24 have S25: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B ≠ 0 ⟶
 ( ( A / B ) = C ⟷ ( B · C ) = A ) ) by (rule MMI_dedth3h)
   from S25 show ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ ( B · C ) = A ) by (rule MMI_imp)
qed
```

```
lemma (in MMIsar0) MMI_divmul2t:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ A = ( B · C ) )
proof -
   have S1: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ ( B · C ) = A ) by (rule MMI_divmult)
   have S2: ( B · C ) = A ⟷ A = ( B · C ) by (rule MMI_eqcom)
   from S1 S2 show ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ A = ( B · C ) ) by (rule MMI_syl6bb)
qed

lemma (in MMIsar0) MMI_divmul3t:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ A = ( C · B ) )
proof -
   have S1: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ A = ( B · C ) ) by (rule MMI_divmul2t)
   have S2: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( B · C ) = ( C · B ) by (rule MMI_axmulcom)
   from S2 have S3: ( B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A = ( B · C ) ⟷ A = ( C · B ) ) by (rule MMI_eqeq2d)
   from S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ⟶
 ( A = ( B · C ) ⟷ A = ( C · B ) ) by (rule MMI_3adant1)
   from S4 have S5: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( A = ( B · C ) ⟷ A = ( C · B ) ) by (rule MMI_adantr)
   from S1 S5 show ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = C ⟷ A = ( C · B ) ) by (rule MMI_bitrd)
qed

lemma (in MMIsar0) MMI_divcl: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: B ≠ 0
   shows ( A / B ) ∈ ℂ
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A3 have S3: B ≠ 0.
   from S1 S2 S3 have S4: ( A / B ) =
 ⋃ { x ∈ ℂ . ( B · x ) = A } by (rule MMI_divval)
   from A2 have S5: B ∈ ℂ.
   from A1 have S6: A ∈ ℂ.
   from A3 have S7: B ≠ 0.
   from S5 S6 S7 have S8: ∃! x . x ∈ ℂ ∧ ( B · x ) = A by (rule MMI_receu)
   have S9: ( ∃! x . x ∈ ℂ ∧ ( B · x ) =
 A ) ⟶ ⋃ { x ∈ ℂ . ( B · x ) = A } ∈ ℂ by (rule MMI_reucl)
   from S8 S9 have S10: ⋃ { x ∈ ℂ . ( B · x ) = A } ∈ ℂ by (rule MMI_ax_mp)
   from S4 S10 show ( A / B ) ∈ ℂ by (rule MMI_eqeltr)
qed
```

lemma (**in** MMIsar0) MMI_divclz: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows B ≠ **0** ⟶ (A / B) ∈ ℂ
proof -
  have S1: B =
 if (B ≠ **0** , B , **1**) ⟶
 (A / B) =
 (A / if (B ≠ **0** , B , **1**)) **by** (rule MMI_opreq2)
  from S1 **have** S2: B =
 if (B ≠ **0** , B , **1**) ⟶
 ((A / B) ∈ ℂ ⟷
 (A / if (B ≠ **0** , B , **1**)) ∈ ℂ) **by** (rule MMI_eleq1d)
  from A1 **have** S3: A ∈ ℂ.
  from A2 **have** S4: B ∈ ℂ.
  have S5: **1** ∈ ℂ **by** (rule MMI_1cn)
  from S4 S5 **have** S6: if (B ≠ **0** , B , **1**) ∈ ℂ **by** (rule MMI_keepel)
  have S7: if (B ≠ **0** , B , **1**) ≠ **0** **by** (rule MMI_elimne0)
  from S3 S6 S7 **have** S8: (A / if (B ≠ **0** , B , **1**)) ∈ ℂ **by** (rule
MMI_divcl)
  from S2 S8 **show** B ≠ **0** ⟶ (A / B) ∈ ℂ **by** (rule MMI_dedth)
qed

lemma (**in** MMIsar0) MMI_divclt:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
 (A / B) ∈ ℂ
proof -
  have S1: A =
 if (A ∈ ℂ , A , **0**) ⟶
 (A / B) =
 (if (A ∈ ℂ , A , **0**) / B) **by** (rule MMI_opreq1)
  from S1 **have** S2: A =
 if (A ∈ ℂ , A , **0**) ⟶
 ((A / B) ∈ ℂ ⟷
 (if (A ∈ ℂ , A , **0**) / B) ∈ ℂ) **by** (rule MMI_eleq1d)
  from S2 **have** S3: A =
 if (A ∈ ℂ , A , **0**) ⟶
 ((B ≠ **0** ⟶ (A / B) ∈ ℂ) ⟷
 (B ≠ **0** ⟶
 (if (A ∈ ℂ , A , **0**) / B) ∈ ℂ)) **by** (rule MMI_imbi2d)
  have S4: B =
 if (B ∈ ℂ , B , **0**) ⟶
 (B ≠ **0** ⟷ if (B ∈ ℂ , B , **0**) ≠ **0**) **by** (rule MMI_neeq1)
  have S5: B =
 if (B ∈ ℂ , B , **0**) ⟶
 (if (A ∈ ℂ , A , **0**) / B) =

```
 ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) by (rule MMI_opreq2)
   from S5 have S6: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( if ( A ∈ ℂ , A , 0 ) / B ) ∈ ℂ ⟷
 ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) ∈ ℂ ) by (rule MMI_eleq1d)
   from S4 S6 have S7: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( B ≠ 0 ⟶ ( if ( A ∈ ℂ , A , 0 ) / B ) ∈ ℂ ) ⟷
 ( if ( B ∈ ℂ , B , 0 ) ≠ 0 ⟶
 ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) ∈ ℂ ) ) by (rule MMI_imbi12d)
   have S8: 0 ∈ ℂ by (rule MMI_0cn)
   from S8 have S9: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   have S10: 0 ∈ ℂ by (rule MMI_0cn)
   from S10 have S11: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S9 S11 have S12: if ( B ∈ ℂ , B , 0 ) ≠ 0 ⟶
 ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) ∈ ℂ by (rule MMI_divclz)
   from S3 S7 S12 have S13: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( B ≠ 0 ⟶ ( A / B ) ∈ ℂ ) by (rule MMI_dedth2h)
   from S13 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) ∈ ℂ by (rule MMI_3impia)
qed

lemma (in MMIsar0) MMI_reccl: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ( 1 / A ) ∈ ℂ
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S2: A ∈ ℂ.
   from A2 have S3: A ≠ 0.
   from S1 S2 S3 show ( 1 / A ) ∈ ℂ by (rule MMI_divcl)
qed

lemma (in MMIsar0) MMI_recclz: assumes A1: A ∈ ℂ
   shows A ≠ 0 ⟶ ( 1 / A ) ∈ ℂ
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S2: A ∈ ℂ.
   from S1 S2 show A ≠ 0 ⟶ ( 1 / A ) ∈ ℂ by (rule MMI_divclz)
qed

lemma (in MMIsar0) MMI_recclt:
   shows ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 1 / A ) ∈ ℂ
proof -
   have S1: 1 ∈ ℂ by (rule MMI_1cn)
   have S2: ( 1 ∈ ℂ ∧ A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( 1 / A ) ∈ ℂ by (rule MMI_divclt)
   from S1 S2 show ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 1 / A ) ∈ ℂ by (rule MMI_mp3an1)
qed
```

```
lemma (in MMIsar0) MMI_divcan2: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: A ≠ 0
   shows ( A · ( B / A ) ) = B
proof -
   have S1: ( B / A ) = ( B / A ) by (rule MMI_eqid)
   from A2 have S2: B ∈ ℂ.
   from A1 have S3: A ∈ ℂ.
   from A2 have S4: B ∈ ℂ.
   from A1 have S5: A ∈ ℂ.
   from A3 have S6: A ≠ 0.
   from S4 S5 S6 have S7: ( B / A ) ∈ ℂ by (rule MMI_divcl)
   from A3 have S8: A ≠ 0.
   from S2 S3 S7 S8 have S9: ( B / A ) =
 ( B / A ) ⟷ ( A · ( B / A ) ) = B by (rule MMI_divmul)
   from S1 S9 show ( A · ( B / A ) ) = B by (rule MMI_mpbi)
qed

lemma (in MMIsar0) MMI_divcan1: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: A ≠ 0
   shows ( ( B / A ) · A ) = B
proof -
   from A2 have S1: B ∈ ℂ.
   from A1 have S2: A ∈ ℂ.
   from A3 have S3: A ≠ 0.
   from S1 S2 S3 have S4: ( B / A ) ∈ ℂ by (rule MMI_divcl)
   from A1 have S5: A ∈ ℂ.
   from S4 S5 have S6: ( ( B / A ) · A ) = ( A · ( B / A ) ) by (rule
MMI_mulcom)
   from A1 have S7: A ∈ ℂ.
   from A2 have S8: B ∈ ℂ.
   from A3 have S9: A ≠ 0.
   from S7 S8 S9 have S10: ( A · ( B / A ) ) = B by (rule MMI_divcan2)
   from S6 S10 show ( ( B / A ) · A ) = B by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_divcan1z: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows A ≠ 0 ⟶ ( ( B / A ) · A ) = B
proof -
   have S1: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 ( B / A ) =
 ( B / if ( A ≠ 0 , A , 1 ) ) by (rule MMI_opreq2)
   have S2: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 A = if ( A ≠ 0 , A , 1 ) by (rule MMI_id)
   from S1 S2 have S3: A =
```

if (A $\neq$ **0** , A , **1**) $\longrightarrow$
((B / A) · A) =
((B / if (A $\neq$ **0** , A , **1**)) · if (A $\neq$ **0** , A , **1**)) **by** (rule MMI_opreq12d)
from S3 **have** S4: A =
if (A $\neq$ **0** , A , **1**) $\longrightarrow$
(((B / A) · A) =
B $\longleftrightarrow$
((B / if (A $\neq$ **0** , A , **1**)) · if (A $\neq$ **0** , A , **1**)) =
B) **by** (rule MMI_eqeq1d)
from A1 **have** S5: A $\in$ $\mathbb{C}$.
have S6: **1** $\in$ $\mathbb{C}$ **by** (rule MMI_1cn)
from S5 S6 **have** S7: if (A $\neq$ **0** , A , **1**) $\in$ $\mathbb{C}$ **by** (rule MMI_keepel)
from A2 **have** S8: B $\in$ $\mathbb{C}$.
have S9: if (A $\neq$ **0** , A , **1**) $\neq$ **0** **by** (rule MMI_elimne0)
from S7 S8 S9 **have** S10: ((B / if (A $\neq$ **0** , A , **1**)) · if (A $\neq$
0 , A , **1**)) =
B **by** (rule MMI_divcan1)
from S4 S10 **show** A $\neq$ **0** $\longrightarrow$ ((B / A) · A) = B **by** (rule MMI_dedth)
qed

lemma (**in** MMIsar0) MMI_divcan2z: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$
shows A $\neq$ **0** $\longrightarrow$ (A · (B / A)) = B
proof -
have S1: A =
if (A $\neq$ **0** , A , **1**) $\longrightarrow$
A = if (A $\neq$ **0** , A , **1**) **by** (rule MMI_id)
have S2: A =
if (A $\neq$ **0** , A , **1**) $\longrightarrow$
(B / A) =
(B / if (A $\neq$ **0** , A , **1**)) **by** (rule MMI_opreq2)
from S1 S2 **have** S3: A =
if (A $\neq$ **0** , A , **1**) $\longrightarrow$
(A · (B / A)) =
(if (A $\neq$ **0** , A , **1**) · (B / if (A $\neq$ **0** , A , **1**))) **by** (rule MMI_opreq12d)
from S3 **have** S4: A =
if (A $\neq$ **0** , A , **1**) $\longrightarrow$
((A · (B / A)) =
B $\longleftrightarrow$
(if (A $\neq$ **0** , A , **1**) · (B / if (A $\neq$ **0** , A , **1**))) =
B) **by** (rule MMI_eqeq1d)
from A1 **have** S5: A $\in$ $\mathbb{C}$.
have S6: **1** $\in$ $\mathbb{C}$ **by** (rule MMI_1cn)
from S5 S6 **have** S7: if (A $\neq$ **0** , A , **1**) $\in$ $\mathbb{C}$ **by** (rule MMI_keepel)
from A2 **have** S8: B $\in$ $\mathbb{C}$.
have S9: if (A $\neq$ **0** , A , **1**) $\neq$ **0** **by** (rule MMI_elimne0)
from S7 S8 S9 **have** S10: (if (A $\neq$ **0** , A , **1**) · (B / if (A $\neq$ **0**
, A , **1**))) =
B **by** (rule MMI_divcan2)

```
   from S4 S10 show A ≠ 0 ⟶ ( A · ( B / A ) ) = B by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_divcan1t:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( ( B / A ) · A ) = B
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A ≠ 0 ⟷ if ( A ∈ ℂ , A , 0 ) ≠ 0 ) by (rule MMI_neeq1)
   have S2: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( B / A ) =
 ( B / if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_opreq2)
   have S3: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 A = if ( A ∈ ℂ , A , 0 ) by (rule MMI_id)
   from S2 S3 have S4: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( B / A ) · A ) =
 ( ( B / if ( A ∈ ℂ , A , 0 ) ) · if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_opreq12d)
   from S4 have S5: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( ( B / A ) · A ) =
 B ⟷
 ( ( B / if ( A ∈ ℂ , A , 0 ) ) · if ( A ∈ ℂ , A , 0 ) ) =
 B ) by (rule MMI_eqeq1d)
   from S1 S5 have S6: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( A ≠ 0 ⟶ ( ( B / A ) · A ) = B ) ⟷
 ( if ( A ∈ ℂ , A , 0 ) ≠ 0 ⟶
 ( ( B / if ( A ∈ ℂ , A , 0 ) ) · if ( A ∈ ℂ , A , 0 ) ) =
 B ) ) by (rule MMI_imbi12d)
   have S7: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( B / if ( A ∈ ℂ , A , 0 ) ) =
 ( if ( B ∈ ℂ , B , 0 ) / if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_opreq1)
   from S7 have S8: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( B / if ( A ∈ ℂ , A , 0 ) ) · if ( A ∈ ℂ , A , 0 ) ) =
 ( ( if ( B ∈ ℂ , B , 0 ) / if ( A ∈ ℂ , A , 0 ) ) · if ( A ∈ ℂ , A ,
0 ) ) by (rule MMI_opreq1d)
   have S9: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 B = if ( B ∈ ℂ , B , 0 ) by (rule MMI_id)
   from S8 S9 have S10: B =
 if ( B ∈ ℂ , B , 0 ) ⟶
 ( ( ( B / if ( A ∈ ℂ , A , 0 ) ) · if ( A ∈ ℂ , A , 0 ) ) =
 B ⟷
```

((if (B ∈ $\mathbb{C}$, B , **0**) / if (A ∈ $\mathbb{C}$, A , **0**)) · if (A ∈ $\mathbb{C}$, A ,
0)) =
if (B ∈ $\mathbb{C}$, B , **0**)) **by** (rule MMI_eqeq12d)
from S10 **have** S11: B =
if (B ∈ $\mathbb{C}$, B , **0**) ⟶
((if (A ∈ $\mathbb{C}$, A , **0**) ≠ **0** ⟶ ((B / if (A ∈ $\mathbb{C}$, A , **0**)) · if
(A ∈ $\mathbb{C}$, A , **0**)) = B) ⟷
(if (A ∈ $\mathbb{C}$, A , **0**) ≠ **0** ⟶
((if (B ∈ $\mathbb{C}$, B , **0**) / if (A ∈ $\mathbb{C}$, A , **0**)) · if (A ∈ $\mathbb{C}$, A ,
0)) =
if (B ∈ $\mathbb{C}$, B , **0**))) **by** (rule MMI_imbi2d)
have S12: **0** ∈ $\mathbb{C}$ **by** (rule MMI_0cn)
from S12 **have** S13: if (A ∈ $\mathbb{C}$, A , **0**) ∈ $\mathbb{C}$ **by** (rule MMI_elimel)
have S14: **0** ∈ $\mathbb{C}$ **by** (rule MMI_0cn)
from S14 **have** S15: if (B ∈ $\mathbb{C}$, B , **0**) ∈ $\mathbb{C}$ **by** (rule MMI_elimel)
from S13 S15 **have** S16: if (A ∈ $\mathbb{C}$, A , **0**) ≠ **0** ⟶
((if (B ∈ $\mathbb{C}$, B , **0**) / if (A ∈ $\mathbb{C}$, A , **0**)) · if (A ∈ $\mathbb{C}$, A ,
0)) =
if (B ∈ $\mathbb{C}$, B , **0**) **by** (rule MMI_divcan1z)
from S6 S11 S16 **have** S17: (A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$) ⟶
(A ≠ **0** ⟶ ((B / A) · A) = B) **by** (rule MMI_dedth2h)
from S17 **show** (A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ A ≠ **0**) ⟶
((B / A) · A) = B **by** (rule MMI_3impia)
qed

lemma (**in** MMIsar0) MMI_divcan2t:
shows (A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ A ≠ **0**) ⟶
(A · (B / A)) = B
proof -
have S1: A =
if (A ∈ $\mathbb{C}$, A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ $\mathbb{C}$, A , **0**) ≠ **0**) **by** (rule MMI_neeq1)
have S2: A =
if (A ∈ $\mathbb{C}$, A , **0**) ⟶
A = if (A ∈ $\mathbb{C}$, A , **0**) **by** (rule MMI_id)
have S3: A =
if (A ∈ $\mathbb{C}$, A , **0**) ⟶
(B / A) =
(B / if (A ∈ $\mathbb{C}$, A , **0**)) **by** (rule MMI_opreq2)
from S2 S3 **have** S4: A =
if (A ∈ $\mathbb{C}$, A , **0**) ⟶
(A · (B / A)) =
(if (A ∈ $\mathbb{C}$, A , **0**) · (B / if (A ∈ $\mathbb{C}$, A , **0**))) **by** (rule MMI_opreq12d)
from S4 **have** S5: A =
if (A ∈ $\mathbb{C}$, A , **0**) ⟶
((A · (B / A)) =
B ⟷
(if (A ∈ $\mathbb{C}$, A , **0**) · (B / if (A ∈ $\mathbb{C}$, A , **0**))) =
B) **by** (rule MMI_eqeq1d)

from S1 S5 **have** S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ (A · (B / A)) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
B)) **by** (rule MMI_imbi12d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(B / if (A ∈ ℂ , A , **0**)) =
(if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**)) **by** (rule MMI_opreq1)
from S7 **have** S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**
))) **by** (rule MMI_opreq2d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
B = if (B ∈ ℂ , B , **0**) **by** (rule MMI_id)
from S8 S9 **have** S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) · (B / if (A ∈ ℂ , A , **0**))) =
B ⟷
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**
))) =
if (B ∈ ℂ , B , **0**)) **by** (rule MMI_eqeq12d)
from S10 **have** S11: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) ≠ **0** ⟶ (if (A ∈ ℂ , A , **0**) · (B / if (
A ∈ ℂ , A , **0**))) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**
))) =
if (B ∈ ℂ , B , **0**))) **by** (rule MMI_imbi2d)
have S12: **0** ∈ ℂ **by** (rule MMI_0cn)
from S12 **have** S13: if (A ∈ ℂ , A , **0**) ∈ ℂ **by** (rule MMI_elimel)
have S14: **0** ∈ ℂ **by** (rule MMI_0cn)
from S14 **have** S15: if (B ∈ ℂ , B , **0**) ∈ ℂ **by** (rule MMI_elimel)
from S13 S15 **have** S16: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) · (if (B ∈ ℂ , B , **0**) / if (A ∈ ℂ , A , **0**
))) =
if (B ∈ ℂ , B , **0**) **by** (rule MMI_divcan2z)
from S6 S11 S16 **have** S17: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A ≠ **0** ⟶ (A · (B / A)) = B) **by** (rule MMI_dedth2h)
from S17 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
(A · (B / A)) = B **by** (rule MMI_3impia)
qed

```
lemma (in MMIsar0) MMI_divne0bt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A ≠ 0 ⟷ ( A / B ) ≠ 0 )
proof -
   have S1: B ∈ ℂ ⟶ ( B · 0 ) = 0 by (rule MMI_mul01t)
   from S1 have S2: B ∈ ℂ ⟶ ( ( B · 0 ) = A ⟷ 0 = A ) by (rule MMI_eqeq1d)
   have S3: A = 0 ⟷ 0 = A by (rule MMI_eqcom)
   from S2 S3 have S4: B ∈ ℂ ⟶ ( A = 0 ⟷ ( B · 0 ) = A ) by (rule
MMI_syl6rbbrA)
   from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A = 0 ⟷ ( B · 0 ) = A ) by (rule MMI_3ad2ant2)
   have S6: 0 ∈ ℂ by (rule MMI_0cn)
   have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ 0 ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = 0 ⟷ ( B · 0 ) = A ) by (rule MMI_divmult)
   from S6 S7 have S8: ( ( A ∈ ℂ ∧ B ∈ ℂ ) ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = 0 ⟷ ( B · 0 ) = A ) by (rule MMI_mp3anl3)
   from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( ( A / B ) = 0 ⟷ ( B · 0 ) = A ) by (rule MMI_3impa)
   from S5 S9 have S10: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A = 0 ⟷ ( A / B ) = 0 ) by (rule MMI_bitr4d)
   from S10 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A ≠ 0 ⟷ ( A / B ) ≠ 0 ) by (rule MMI_eqneqd)
qed

lemma (in MMIsar0) MMI_divne0: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: A ≠ 0 and
    A4: B ≠ 0
   shows ( A / B ) ≠ 0
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A4 have S3: B ≠ 0.
   from A3 have S4: A ≠ 0.
   have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A ≠ 0 ⟷ ( A / B ) ≠ 0 ) by (rule MMI_divne0bt)
   from S4 S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) ≠ 0 by (rule MMI_mpbii)
   from S1 S2 S3 S6 show ( A / B ) ≠ 0 by (rule MMI_mp3an)
qed

lemma (in MMIsar0) MMI_recne0z: assumes A1: A ∈ ℂ
   shows A ≠ 0 ⟶ ( 1 / A ) ≠ 0
proof -
   have S1: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 ( 1 / A ) =
 ( 1 / if ( A ≠ 0 , A , 1 ) ) by (rule MMI_opreq2)
   from S1 have S2: A =
```

```
 if ( A ≠ 0 , A , 1 ) ⟶
 ( ( 1 / A ) ≠ 0 ⟷
 ( 1 / if ( A ≠ 0 , A , 1 ) ) ≠ 0 ) by (rule MMI_neeq1d)
   have S3: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S4: A ∈ ℂ.
   have S5: 1 ∈ ℂ by (rule MMI_1cn)
   from S4 S5 have S6: if ( A ≠ 0 , A , 1 ) ∈ ℂ by (rule MMI_keepel)
   have S7: 1 ≠ 0 by (rule MMI_ax1ne0)
   have S8: if ( A ≠ 0 , A , 1 ) ≠ 0 by (rule MMI_elimne0)
   from S3 S6 S7 S8 have S9: ( 1 / if ( A ≠ 0 , A , 1 ) ) ≠ 0 by (rule
MMI_divne0)
   from S2 S9 show A ≠ 0 ⟶ ( 1 / A ) ≠ 0 by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_recne0t:
   shows ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 1 / A ) ≠ 0
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A ≠ 0 ⟷ if ( A ∈ ℂ , A , 0 ) ≠ 0 ) by (rule MMI_neeq1)
   have S2: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( 1 / A ) =
 ( 1 / if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_opreq2)
   from S2 have S3: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( 1 / A ) ≠ 0 ⟷
 ( 1 / if ( A ∈ ℂ , A , 0 ) ) ≠ 0 ) by (rule MMI_neeq1d)
   from S1 S3 have S4: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( A ≠ 0 ⟶ ( 1 / A ) ≠ 0 ) ⟷
 ( if ( A ∈ ℂ , A , 0 ) ≠ 0 ⟶
 ( 1 / if ( A ∈ ℂ , A , 0 ) ) ≠ 0 ) ) by (rule MMI_imbi12d)
   have S5: 0 ∈ ℂ by (rule MMI_0cn)
   from S5 have S6: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S6 have S7: if ( A ∈ ℂ , A , 0 ) ≠ 0 ⟶
 ( 1 / if ( A ∈ ℂ , A , 0 ) ) ≠ 0 by (rule MMI_recne0z)
   from S4 S7 have S8: A ∈ ℂ ⟶ ( A ≠ 0 ⟶ ( 1 / A ) ≠ 0 ) by (rule
MMI_dedth)
   from S8 show ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 1 / A ) ≠ 0 by (rule MMI_imp)
qed

lemma (in MMIsar0) MMI_recid: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ( A · ( 1 / A ) ) = 1
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: 1 ∈ ℂ by (rule MMI_1cn)
   from A2 have S3: A ≠ 0.
```

```
   from S1 S2 S3 show ( A · ( 1 / A ) ) = 1 by (rule MMI_divcan2)
qed

lemma (in MMIsar0) MMI_recidz: assumes A1: A ∈ ℂ
   shows A ≠ 0 ⟶ ( A · ( 1 / A ) ) = 1
proof -
   from A1 have S1: A ∈ ℂ.
   have S2: 1 ∈ ℂ by (rule MMI_1cn)
   from S1 S2 show A ≠ 0 ⟶ ( A · ( 1 / A ) ) = 1 by (rule MMI_divcan2z)
qed

lemma (in MMIsar0) MMI_recidt:
   shows ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( A · ( 1 / A ) ) = 1
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A ≠ 0 ⟷ if ( A ∈ ℂ , A , 0 ) ≠ 0 ) by (rule MMI_neeq1)
   have S2: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 A = if ( A ∈ ℂ , A , 0 ) by (rule MMI_id)
   have S3: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( 1 / A ) =
 ( 1 / if ( A ∈ ℂ , A , 0 ) ) by (rule MMI_opreq2)
   from S2 S3 have S4: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( A · ( 1 / A ) ) =
 ( if ( A ∈ ℂ , A , 0 ) · ( 1 / if ( A ∈ ℂ , A , 0 ) ) ) by (rule MMI_opreq12d)
   from S4 have S5: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( A · ( 1 / A ) ) =
 1 ⟷
 ( if ( A ∈ ℂ , A , 0 ) · ( 1 / if ( A ∈ ℂ , A , 0 ) ) ) =
 1 ) by (rule MMI_eqeq1d)
   from S1 S5 have S6: A =
 if ( A ∈ ℂ , A , 0 ) ⟶
 ( ( A ≠ 0 ⟶ ( A · ( 1 / A ) ) = 1 ) ⟷
 ( if ( A ∈ ℂ , A , 0 ) ≠ 0 ⟶
 ( if ( A ∈ ℂ , A , 0 ) · ( 1 / if ( A ∈ ℂ , A , 0 ) ) ) =
 1 ) ) by (rule MMI_imbi12d)
   have S7: 0 ∈ ℂ by (rule MMI_0cn)
   from S7 have S8: if ( A ∈ ℂ , A , 0 ) ∈ ℂ by (rule MMI_elimel)
   from S8 have S9: if ( A ∈ ℂ , A , 0 ) ≠ 0 ⟶
 ( if ( A ∈ ℂ , A , 0 ) · ( 1 / if ( A ∈ ℂ , A , 0 ) ) ) =
 1 by (rule MMI_recidz)
   from S6 S9 have S10: A ∈ ℂ ⟶
 ( A ≠ 0 ⟶ ( A · ( 1 / A ) ) = 1 ) by (rule MMI_dedth)
   from S10 show ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
```

```
 ( A · ( 1 / A ) ) = 1 by (rule MMI_imp)
qed

lemma (in MMIsar0) MMI_recid2t:
   shows ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( ( 1 / A ) · A ) = 1
proof -
   have S1: ( ( 1 / A ) ∈ ℂ ∧ A ∈ ℂ ) ⟶
 ( ( 1 / A ) · A ) = ( A · ( 1 / A ) ) by (rule MMI_axmulcom)
   have S2: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 1 / A ) ∈ ℂ by (rule MMI_recclt)
   have S3: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ A ∈ ℂ by (rule MMI_pm3_26)
   from S1 S2 S3 have S4: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( ( 1 / A ) · A ) = ( A · ( 1 / A ) ) by (rule MMI_sylanc)
   have S5: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( A · ( 1 / A ) ) = 1 by (rule MMI_recidt)
   from S4 S5 show ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( ( 1 / A ) · A ) = 1 by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_divrec: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: B ≠ 0
   shows ( A / B ) = ( A · ( 1 / B ) )
proof -
   from A2 have S1: B ∈ ℂ.
   from A1 have S2: A ∈ ℂ.
   from A2 have S3: B ∈ ℂ.
   from A3 have S4: B ≠ 0.
   from S3 S4 have S5: ( 1 / B ) ∈ ℂ by (rule MMI_reccl)
   from S2 S5 have S6: ( A · ( 1 / B ) ) ∈ ℂ by (rule MMI_mulcl)
   from S1 S6 have S7: ( B · ( A · ( 1 / B ) ) ) =
 ( ( A · ( 1 / B ) ) · B ) by (rule MMI_mulcom)
   from A1 have S8: A ∈ ℂ.
   from S5 have S9: ( 1 / B ) ∈ ℂ .
   from A2 have S10: B ∈ ℂ.
   from S8 S9 S10 have S11: ( ( A · ( 1 / B ) ) · B ) =
 ( A · ( ( 1 / B ) · B ) ) by (rule MMI_mulass)
   from A2 have S12: B ∈ ℂ.
   have S13: 1 ∈ ℂ by (rule MMI_1cn)
   from A3 have S14: B ≠ 0.
   from S12 S13 S14 have S15: ( ( 1 / B ) · B ) = 1 by (rule MMI_divcan1)
   from S15 have S16: ( A · ( ( 1 / B ) · B ) ) = ( A · 1 ) by (rule MMI_opreq2i)
   from A1 have S17: A ∈ ℂ.
   from S17 have S18: ( A · 1 ) = A by (rule MMI_mulid1)
   from S16 S18 have S19: ( A · ( ( 1 / B ) · B ) ) = A by (rule MMI_eqtr)
   from S7 S11 S19 have S20: ( B · ( A · ( 1 / B ) ) ) = A by (rule MMI_3eqtr)
   from A1 have S21: A ∈ ℂ.
   from A2 have S22: B ∈ ℂ.
   from S6 have S23: ( A · ( 1 / B ) ) ∈ ℂ .
```

from A3 have S24: B $\neq$ **0**.
from S21 S22 S23 S24 have S25: (A / B) =
(A · (**1** / B)) $\longleftrightarrow$
(B · (A · (**1** / B))) = A by (rule MMI_divmul)
from S20 S25 show (A / B) = (A · (**1** / B)) by (rule MMI_mpbir)
qed

lemma (in MMIsar0) MMI_divrecz: assumes A1: A $\in$ $\mathbb{C}$ and
A2: B $\in$ $\mathbb{C}$
shows B $\neq$ **0** $\longrightarrow$ (A / B) = (A · (**1** / B))
proof -
have S1: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(A / B) =
(A / if (B $\neq$ **0** , B , **1**)) by (rule MMI_opreq2)
have S2: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(**1** / B) =
(**1** / if (B $\neq$ **0** , B , **1**)) by (rule MMI_opreq2)
from S2 have S3: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
(A · (**1** / B)) =
(A · (**1** / if (B $\neq$ **0** , B , **1**))) by (rule MMI_opreq2d)
from S1 S3 have S4: B =
if (B $\neq$ **0** , B , **1**) $\longrightarrow$
((A / B) =
(A · (**1** / B)) $\longleftrightarrow$
(A / if (B $\neq$ **0** , B , **1**)) =
(A · (**1** / if (B $\neq$ **0** , B , **1**)))) by (rule MMI_eqeq12d)
from A1 have S5: A $\in$ $\mathbb{C}$.
from A2 have S6: B $\in$ $\mathbb{C}$.
have S7: **1** $\in$ $\mathbb{C}$ by (rule MMI_1cn)
from S6 S7 have S8: if (B $\neq$ **0** , B , **1**) $\in$ $\mathbb{C}$ by (rule MMI_keepel)
have S9: if (B $\neq$ **0** , B , **1**) $\neq$ **0** by (rule MMI_elimne0)
from S5 S8 S9 have S10: (A / if (B $\neq$ **0** , B , **1**)) =
(A · (**1** / if (B $\neq$ **0** , B , **1**))) by (rule MMI_divrec)
from S4 S10 show B $\neq$ **0** $\longrightarrow$ (A / B) = (A · (**1** / B))
by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_divrect:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ B $\neq$ **0**) $\longrightarrow$
(A / B) = (A · (**1** / B))
proof -
have S1: A =
if (A $\in$ $\mathbb{C}$, A , **0**) $\longrightarrow$
(A / B) =

(if (A ∈ ℂ , A , **0**) / B) **by** (rule MMI_opreq1)
  have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · (**1** / B)) =
(if (A ∈ ℂ , A , **0**) · (**1** / B)) **by** (rule MMI_opreq1)
  from S1 S2 **have** S3: A =
if (A ∈ ℂ , A , **0**) ⟶
((A / B) =
(A · (**1** / B)) ⟷
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) · (**1** / B))) **by** (rule MMI_eqeq12d)
  from S3 **have** S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((B ≠ **0** ⟶ (A / B) = (A · (**1** / B))) ⟷
(B ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) · (**1** / B)))) **by** (rule MMI_imbi2d)
  have S5: B =
if (B ∈ ℂ , B , **0**) ⟶
(B ≠ **0** ⟷ if (B ∈ ℂ , B , **0**) ≠ **0**) **by** (rule MMI_neeq1)
  have S6: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) **by** (rule MMI_opreq2)
  have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(**1** / B) =
(**1** / if (B ∈ ℂ , B , **0**)) **by** (rule MMI_opreq2)
  from S7 **have** S8: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · (**1** / B)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**))) **by** (rule MMI_opreq2d)
  from S6 S8 **have** S9: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / B) =
(if (A ∈ ℂ , A , **0**) · (**1** / B)) ⟷
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**)))) **by** (rule MMI_eqeq12d)
  from S5 S9 **have** S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((B ≠ **0** ⟶ (if (A ∈ ℂ , A , **0**) / B) = (if (A ∈ ℂ , A , **0**)
· (**1** / B))) ⟷
(if (B ∈ ℂ , B , **0**) ≠ **0** ⟶
(if (A ∈ ℂ , A , **0**) / if (B ∈ ℂ , B , **0**)) =
(if (A ∈ ℂ , A , **0**) · (**1** / if (B ∈ ℂ , B , **0**))))) **by** (rule
MMI_imbi12d)
  have S11: **0** ∈ ℂ **by** (rule MMI_0cn)
  from S11 **have** S12: if (A ∈ ℂ , A , **0**) ∈ ℂ **by** (rule MMI_elimel)
  have S13: **0** ∈ ℂ **by** (rule MMI_0cn)

```
    from S13 have S14: if ( B ∈ ℂ , B , 0 ) ∈ ℂ by (rule MMI_elimel)
    from S12 S14 have S15: if ( B ∈ ℂ , B , 0 ) ≠ 0 ⟶
 ( if ( A ∈ ℂ , A , 0 ) / if ( B ∈ ℂ , B , 0 ) ) =
 ( if ( A ∈ ℂ , A , 0 ) · ( 1 / if ( B ∈ ℂ , B , 0 ) ) ) by (rule MMI_divrecz)
    from S4 S10 S15 have S16: ( A ∈ ℂ ∧ B ∈ ℂ ) ⟶
 ( B ≠ 0 ⟶
 ( A / B ) = ( A · ( 1 / B ) ) ) by (rule MMI_dedth2h)
    from S16 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) = ( A · ( 1 / B ) ) by (rule MMI_3impia)
qed

lemma (in MMIsar0) MMI_divrec2t:
    shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) = ( ( 1 / B ) · A )
proof -
    have S1: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) = ( A · ( 1 / B ) ) by (rule MMI_divrect)
    have S2: ( A ∈ ℂ ∧ ( 1 / B ) ∈ ℂ ) ⟶
 ( A · ( 1 / B ) ) = ( ( 1 / B ) · A ) by (rule MMI_axmulcom)
    have S3: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶ A ∈ ℂ by (rule MMI_3simp1)
    have S4: ( B ∈ ℂ ∧ B ≠ 0 ) ⟶ ( 1 / B ) ∈ ℂ by (rule MMI_recclt)
    from S4 have S5: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( 1 / B ) ∈ ℂ by (rule MMI_3adant1)
    from S2 S3 S5 have S6: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A · ( 1 / B ) ) = ( ( 1 / B ) · A ) by (rule MMI_sylanc)
    from S1 S6 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) = ( ( 1 / B ) · A ) by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_divasst:
    shows ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( ( A · B ) / C ) = ( A · ( B / C ) )
proof -
    have S1: A ∈ ℂ ⟶ A ∈ ℂ by (rule MMI_id)
    have S2: B ∈ ℂ ⟶ B ∈ ℂ by (rule MMI_id)
    have S3: ( C ∈ ℂ ∧ C ≠ 0 ) ⟶ ( 1 / C ) ∈ ℂ by (rule MMI_recclt)
    from S1 S2 S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( C ∈ ℂ ∧ C ≠ 0 ) ) ⟶

 ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( 1 / C ) ∈ ℂ ) by (rule MMI_3anim123i)
    from S4 have S5: A ∈ ℂ ⟶
 ( B ∈ ℂ ⟶
 ( ( C ∈ ℂ ∧ C ≠ 0 ) ⟶
 ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( 1 / C ) ∈ ℂ ) ) ) by (rule MMI_3exp)
    from S5 have S6: A ∈ ℂ ⟶
 ( B ∈ ℂ ⟶
 ( C ∈ ℂ ⟶
 ( C ≠ 0 ⟶
 ( A ∈ ℂ ∧ B ∈ ℂ ∧ ( 1 / C ) ∈ ℂ ) ) ) ) by (rule MMI_exp4a)
    from S6 have S7: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
```

(A ∈ ℂ ∧ B ∈ ℂ ∧ (**1** / C) ∈ ℂ) **by** (rule MMI_3imp1)
have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ (**1** / C) ∈ ℂ) ⟶
((A · B) · (**1** / C)) =
(A · (B · (**1** / C))) **by** (rule MMI_axmulass)
from S7 S8 **have** S9: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) · (**1** / C)) =
(A · (B · (**1** / C))) **by** (rule MMI_syl)
have S10: ((A · B) ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A · B) / C) =
((A · B) · (**1** / C)) **by** (rule MMI_divrect)
from S10 **have** S11: (((A · B) ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) =
((A · B) · (**1** / C)) **by** (rule MMI_3expa)
have S12: (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A · B) ∈ ℂ **by** (rule MMI_axmulcl)
from S12 **have** S13: ((A ∈ ℂ ∧ B ∈ ℂ) ∧ C ∈ ℂ) ⟶
((A · B) ∈ ℂ ∧ C ∈ ℂ) **by** (rule MMI_anim1i)
from S13 **have** S14: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A · B) ∈ ℂ ∧ C ∈ ℂ) **by** (rule MMI_3impa)
from S11 S14 **have** S15: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

((A · B) / C) =
((A · B) · (**1** / C)) **by** (rule MMI_sylan)
have S16: (B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(B / C) = (B · (**1** / C)) **by** (rule MMI_divrect)
from S16 **have** S17: ((B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) = (B · (**1** / C)) **by** (rule MMI_3expa)
from S17 **have** S18: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) = (B · (**1** / C)) **by** (rule MMI_3adantl1)
from S18 **have** S19: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · (B / C)) =
(A · (B · (**1** / C))) **by** (rule MMI_opreq2d)
from S9 S15 S19 **show** ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

((A · B) / C) = (A · (B / C)) **by** (rule MMI_3eqtr4d)
qed

lemma (**in** MMIsar0) MMI_div23t:
shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = ((A / C) · B)
proof -
have S1: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A · B) = (B · A) **by** (rule MMI_axmulcom)
from S1 **have** S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A · B) = (B · A) **by** (rule MMI_3adant3)
from S2 **have** S3: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · B) = (B · A) **by** (rule MMI_adantr)
from S3 **have** S4: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A · B) / C) = ((B · A) / C) **by** (rule MMI_opreq1d)
have S5: ((B ∈ ℂ ∧ A ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

((B · A) / C) = (B · (A / C)) **by** (rule MMI_divasst)
  from S5 **have** S6: (B $\in$ $\mathbb{C}$ $\wedge$ A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(C $\neq$ **0** $\longrightarrow$
((B · A) / C) =
(B · (A / C))) **by** (rule MMI_ex)
  from S6 **have** S7: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(C $\neq$ **0** $\longrightarrow$
((B · A) / C) =
(B · (A / C))) **by** (rule MMI_3com12)
  from S7 **have** S8: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((B · A) / C) = (B · (A / C)) **by** (rule MMI_imp)
  have S9: (B $\in$ $\mathbb{C}$ $\wedge$ (A / C) $\in$ $\mathbb{C}$) $\longrightarrow$
(B · (A / C)) = ((A / C) · B) **by** (rule MMI_axmulcom)
  have S10: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$ B $\in$ $\mathbb{C}$ **by** (rule MMI_3simp2)
  from S10 **have** S11: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
B $\in$ $\mathbb{C}$ **by** (rule MMI_adantr)
  have S12: (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$ $\wedge$ C $\neq$ **0**) $\longrightarrow$
(A / C) $\in$ $\mathbb{C}$ **by** (rule MMI_divclt)
  from S12 **have** S13: ((A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
(A / C) $\in$ $\mathbb{C}$ **by** (rule MMI_3expa)
  from S13 **have** S14: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
(A / C) $\in$ $\mathbb{C}$ **by** (rule MMI_3adantl2)
  from S9 S11 S14 **have** S15: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**)
$\longrightarrow$
(B · (A / C)) = ((A / C) · B) **by** (rule MMI_sylanc)
  from S4 S8 S15 **show** ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A · B) / C) = ((A / C) · B) **by** (rule MMI_3eqtrd)
qed

lemma (**in** MMIsar0) MMI_div13t:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ B $\neq$ **0**) $\longrightarrow$
((A / B) · C) = ((C / B) · A)
proof -
  have S1: (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A · C) = (C · A) **by** (rule MMI_axmulcom)
  from S1 **have** S2: (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A · C) / B) = ((C · A) / B) **by** (rule MMI_opreq1d)
  from S2 **have** S3: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A · C) / B) = ((C · A) / B) **by** (rule MMI_3adant2)
  from S3 **have** S4: ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ B $\neq$ **0**) $\longrightarrow$
((A · C) / B) = ((C · A) / B) **by** (rule MMI_adantr)
  have S5: ((A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ B $\neq$ **0**) $\longrightarrow$
((A · C) / B) = ((A / B) · C) **by** (rule MMI_div23t)
  from S5 **have** S6: (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
(B $\neq$ **0** $\longrightarrow$
((A · C) / B) =
((A / B) · C)) **by** (rule MMI_ex)
  from S6 **have** S7: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(B $\neq$ **0** $\longrightarrow$

((A · C) / B) =
((A / B) · C)) **by** (rule MMI_3com23)
  from S7 **have** S8: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A · C) / B) = ((A / B) · C) **by** (rule MMI_imp)
  have S9: ((C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ∧ B ≠ **0**) ⟶
((C · A) / B) = ((C / B) · A) **by** (rule MMI_div23t)
  from S9 **have** S10: (C ∈ ℂ ∧ A ∈ ℂ ∧ B ∈ ℂ) ⟶
(B ≠ **0** ⟶
((C · A) / B) =
((C / B) · A)) **by** (rule MMI_ex)
  from S10 **have** S11: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(B ≠ **0** ⟶
((C · A) / B) =
((C / B) · A)) **by** (rule MMI_3coml)
  from S11 **have** S12: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((C · A) / B) = ((C / B) · A) **by** (rule MMI_imp)
  from S4 S8 S12 **show** ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ B ≠ **0**) ⟶
((A / B) · C) = ((C / B) · A) **by** (rule MMI_3eqtr3d)
qed

lemma **(in** MMIsar0) MMI_div12t:
  shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A · (B / C)) = (B · (A / C))
proof -
  have S1: (A ∈ ℂ ∧ (B / C) ∈ ℂ) ⟶
(A · (B / C)) = ((B / C) · A) **by** (rule MMI_axmulcom)
  have S2: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ A ∈ ℂ **by** (rule MMI_3simp1)
  from S2 **have** S3: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
A ∈ ℂ **by** (rule MMI_adantr)
  have S4: (B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ **by** (rule MMI_divclt)
  from S4 **have** S5: ((B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ **by** (rule MMI_3expa)
  from S5 **have** S6: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ **by** (rule MMI_3adantl1)
  from S1 S3 S6 **have** S7: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

(A · (B / C)) = ((B / C) · A) **by** (rule MMI_sylanc)
  have S8: ((B ∈ ℂ ∧ C ∈ ℂ ∧ A ∈ ℂ) ∧ C ≠ **0**) ⟶
((B / C) · A) = ((A / C) · B) **by** (rule MMI_div13t)
  from S8 **have** S9: (B ∈ ℂ ∧ C ∈ ℂ ∧ A ∈ ℂ) ⟶
(C ≠ **0** ⟶
((B / C) · A) =
((A / C) · B)) **by** (rule MMI_ex)
  from S9 **have** S10: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C ≠ **0** ⟶
((B / C) · A) =
((A / C) · B)) **by** (rule MMI_3comr)
  from S10 **have** S11: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

((B / C) · A) = ((A / C) · B) **by** (rule MMI_imp)
  have S12: ((A / C) ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$) ⟶
((A / C) · B) = (B · (A / C)) **by** (rule MMI_axmulcom)
  have S13: (A ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$ ∧ C ≠ **0**) ⟶
(A / C) ∈ $\mathbb{C}$ **by** (rule MMI_divclt)
  from S13 **have** S14: ((A ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ∧ C ≠ **0**) ⟶
(A / C) ∈ $\mathbb{C}$ **by** (rule MMI_3expa)
  from S14 **have** S15: ((A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ∧ C ≠ **0**) ⟶
(A / C) ∈ $\mathbb{C}$ **by** (rule MMI_3adantl2)
  have S16: (A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ⟶ B ∈ $\mathbb{C}$ **by** (rule MMI_3simp2)
  from S16 **have** S17: ((A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ∧ C ≠ **0**) ⟶
B ∈ $\mathbb{C}$ **by** (rule MMI_adantr)
  from S12 S15 S17 **have** S18: ((A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ∧ C ≠ **0**
) ⟶
((A / C) · B) = (B · (A / C)) **by** (rule MMI_sylanc)
  from S7 S11 S18 **show** ((A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ∧ C ≠ **0**) ⟶

(A · (B / C)) = (B · (A / C)) **by** (rule MMI_3eqtrd)
qed

lemma **(in** MMIsar0) MMI_divassz: **assumes** A1: A ∈ $\mathbb{C}$ **and**
   A2: B ∈ $\mathbb{C}$ **and**
   A3: C ∈ $\mathbb{C}$
  shows C ≠ **0** ⟶
((A · B) / C) = (A · (B / C))
proof -
  from A1 **have** S1: A ∈ $\mathbb{C}$.
  from A2 **have** S2: B ∈ $\mathbb{C}$.
  from A3 **have** S3: C ∈ $\mathbb{C}$.
  from S1 S2 S3 **have** S4: A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$ **by** (rule MMI_3pm3_2i)
  have S5: ((A ∈ $\mathbb{C}$ ∧ B ∈ $\mathbb{C}$ ∧ C ∈ $\mathbb{C}$) ∧ C ≠ **0**) ⟶
((A · B) / C) = (A · (B / C)) **by** (rule MMI_divasst)
  from S4 S5 **show** C ≠ **0** ⟶
((A · B) / C) = (A · (B / C)) **by** (rule MMI_mpan)
qed

lemma **(in** MMIsar0) MMI_divass: **assumes** A1: A ∈ $\mathbb{C}$ **and**
   A2: B ∈ $\mathbb{C}$ **and**
   A3: C ∈ $\mathbb{C}$ **and**
   A4: C ≠ **0**
  shows ((A · B) / C) = (A · (B / C))
proof -
  from A4 **have** S1: C ≠ **0**.
  from A1 **have** S2: A ∈ $\mathbb{C}$.
  from A2 **have** S3: B ∈ $\mathbb{C}$.
  from A3 **have** S4: C ∈ $\mathbb{C}$.
  from S2 S3 S4 **have** S5: C ≠ **0** ⟶
((A · B) / C) = (A · (B / C)) **by** (rule MMI_divassz)
  from S1 S5 **show** ((A · B) / C) = (A · (B / C)) **by** (rule MMI_ax_mp)

qed

lemma (**in** MMIsar0) MMI_divdir: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$ **and**
    A4: C $\neq$ **0**
  shows ((A + B) / C) =
 ((A / C) + (B / C))
proof -
  from A1 **have** S1: A $\in$ $\mathbb{C}$.
  from A2 **have** S2: B $\in$ $\mathbb{C}$.
  from A3 **have** S3: C $\in$ $\mathbb{C}$.
  from A4 **have** S4: C $\neq$ **0**.
  from S3 S4 **have** S5: (**1** / C) $\in$ $\mathbb{C}$ **by** (rule MMI_reccl)
  from S1 S2 S5 **have** S6: ((A + B) $\cdot$ (**1** / C)) =
 ((A $\cdot$ (**1** / C)) + (B $\cdot$ (**1** / C))) **by** (rule MMI_adddir)
  from A1 **have** S7: A $\in$ $\mathbb{C}$.
  from A2 **have** S8: B $\in$ $\mathbb{C}$.
  from S7 S8 **have** S9: (A + B) $\in$ $\mathbb{C}$ **by** (rule MMI_addcl)
  from A3 **have** S10: C $\in$ $\mathbb{C}$.
  from A4 **have** S11: C $\neq$ **0**.
  from S9 S10 S11 **have** S12: ((A + B) / C) =
 ((A + B) $\cdot$ (**1** / C)) **by** (rule MMI_divrec)
  from A1 **have** S13: A $\in$ $\mathbb{C}$.
  from A3 **have** S14: C $\in$ $\mathbb{C}$.
  from A4 **have** S15: C $\neq$ **0**.
  from S13 S14 S15 **have** S16: (A / C) = (A $\cdot$ (**1** / C)) **by** (rule
MMI_divrec)
  from A2 **have** S17: B $\in$ $\mathbb{C}$.
  from A3 **have** S18: C $\in$ $\mathbb{C}$.
  from A4 **have** S19: C $\neq$ **0**.
  from S17 S18 S19 **have** S20: (B / C) = (B $\cdot$ (**1** / C)) **by** (rule
MMI_divrec)
  from S16 S20 **have** S21: ((A / C) + (B / C)) =
 ((A $\cdot$ (**1** / C)) + (B $\cdot$ (**1** / C))) **by** (rule MMI_opreq12i)
  from S6 S12 S21 **show** ((A + B) / C) =
 ((A / C) + (B / C)) **by** (rule MMI_3eqtr4)
qed

lemma (**in** MMIsar0) MMI_div23: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$ **and**
    A4: C $\neq$ **0**
  shows ((A $\cdot$ B) / C) = ((A / C) $\cdot$ B)
proof -
  from A1 **have** S1: A $\in$ $\mathbb{C}$.
  from A2 **have** S2: B $\in$ $\mathbb{C}$.
  from S1 S2 **have** S3: (A $\cdot$ B) = (B $\cdot$ A) **by** (rule MMI_mulcom)
  from S3 **have** S4: ((A $\cdot$ B) / C) = ((B $\cdot$ A) / C)

```
    by (rule MMI_opreq1i)
  from A2 have S5: B ∈ ℂ.
  from A1 have S6: A ∈ ℂ.
  from A3 have S7: C ∈ ℂ.
  from A4 have S8: C ≠ 0.
  from S5 S6 S7 S8 have
    S9: ( ( B · A ) / C ) = ( B · ( A / C ) ) by (rule MMI_divass)
  from A2 have S10: B ∈ ℂ.
  from A1 have S11: A ∈ ℂ.
  from A3 have S12: C ∈ ℂ.
  from A4 have S13: C ≠ 0.
  from S11 S12 S13 have S14: ( A / C ) ∈ ℂ by (rule MMI_divcl)
  from S10 S14 have S15: ( B · ( A / C ) ) = ( ( A / C ) · B )
    by (rule MMI_mulcom)
  from S4 S9 S15 show ( ( A · B ) / C ) = ( ( A / C ) · B )
    by (rule MMI_3eqtr)
qed
```

```
lemma (in MMIsar0) MMI_divdirz: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: C ∈ ℂ
  shows C ≠ 0 ⟶
 ( ( A + B ) / C ) =
 ( ( A / C ) + ( B / C ) )
proof -
  have S1: C =
 if ( C ≠ 0 , C , 1 ) ⟶
 ( ( A + B ) / C ) =
 ( ( A + B ) / if ( C ≠ 0 , C , 1 ) ) by (rule MMI_opreq2)
  have S2: C =
 if ( C ≠ 0 , C , 1 ) ⟶
 ( A / C ) =
 ( A / if ( C ≠ 0 , C , 1 ) ) by (rule MMI_opreq2)
  have S3: C =
 if ( C ≠ 0 , C , 1 ) ⟶
 ( B / C ) =
 ( B / if ( C ≠ 0 , C , 1 ) ) by (rule MMI_opreq2)
  from S2 S3 have S4: C =
 if ( C ≠ 0 , C , 1 ) ⟶
 ( ( A / C ) + ( B / C ) ) =
 ( ( A / if ( C ≠ 0 , C , 1 ) ) + ( B / if ( C ≠ 0 , C , 1 ) ) ) by
(rule MMI_opreq12d)
  from S1 S4 have S5: C =
 if ( C ≠ 0 , C , 1 ) ⟶
 ( ( ( A + B ) / C ) =
 ( ( A / C ) + ( B / C ) ) ⟷
```

((A + B) / if (C ≠ **0** , C , **1**)) =
((A / if (C ≠ **0** , C , **1**)) + (B / if (C ≠ **0** , C , **1**)))) **by**
(rule MMI_eqeq12d)
  from A1 **have** S6: A ∈ ℂ.
  from A2 **have** S7: B ∈ ℂ.
  from A3 **have** S8: C ∈ ℂ.
  have S9: **1** ∈ ℂ **by** (rule MMI_1cn)
  from S8 S9 **have** S10: if (C ≠ **0** , C , **1**) ∈ ℂ **by** (rule MMI_keepel)
  have S11: if (C ≠ **0** , C , **1**) ≠ **0** **by** (rule MMI_elimne0)
  from S6 S7 S10 S11 **have** S12: ((A + B) / if (C ≠ **0** , C , **1**))
=
((A / if (C ≠ **0** , C , **1**)) + (B / if (C ≠ **0** , C , **1**))) **by**
(rule MMI_divdir)
  from S5 S12 **show** C ≠ **0** ⟶
((A + B) / C) =
((A / C) + (B / C)) **by** (rule MMI_dedth)
qed

lemma (**in** MMIsar0) MMI_divdirt:
  shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + B) / C) =
((A / C) + (B / C))
proof -
  have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A + B) =
(if (A ∈ ℂ , A , **0**) + B) **by** (rule MMI_opreq1)
  from S1 **have** S2: A =
if (A ∈ ℂ , A , **0**) ⟶
((A + B) / C) =
((if (A ∈ ℂ , A , **0**) + B) / C) **by** (rule MMI_opreq1d)
  have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
(A / C) =
(if (A ∈ ℂ , A , **0**) / C) **by** (rule MMI_opreq1)
  from S3 **have** S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((A / C) + (B / C)) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C)) **by** (rule MMI_opreq1d)
  from S2 S4 **have** S5: A =
if (A ∈ ℂ , A , **0**) ⟶
(((A + B) / C) =
((A / C) + (B / C)) ⟷
((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C))) **by** (rule MMI_eqeq12d)
  from S5 **have** S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((C ≠ **0** ⟶ ((A + B) / C) = ((A / C) + (B / C))) ⟷
(C ≠ **0** ⟶

((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C)))) **by** (rule MMI_imbi2d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) + B) =
(if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) **by** (rule MMI_opreq2)
from S7 **have** S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) **by** (rule MMI_opreq1d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
(B / C) =
(if (B ∈ ℂ , B , **0**) / C) **by** (rule MMI_opreq1)
from S9 **have** S10: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) / C) + (B / C)) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C)) **by**
(rule MMI_opreq2d)
from S8 S10 **have** S11: B =
if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) + B) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (B / C)) ⟷
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C))) **by**
(rule MMI_eqeq12d)
from S11 **have** S12: B =
if (B ∈ ℂ , B , **0**) ⟶
((C ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) + B) / C) = ((if (A ∈ ℂ
, A , **0**) / C) + (B / C))) ⟷
(C ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) / C) + (if (B ∈ ℂ , B , **0**) / C))))
by (rule MMI_imbi2d)
have S13: C =
if (C ∈ ℂ , C , **0**) ⟶
(C ≠ **0** ⟷ if (C ∈ ℂ , C , **0**) ≠ **0**) **by** (rule MMI_neeq1)
have S14: C =
if (C ∈ ℂ , C , **0**) ⟶
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / C) =
((if (A ∈ ℂ , A , **0**) + if (B ∈ ℂ , B , **0**)) / if (C ∈ ℂ , C
, **0**)) **by** (rule MMI_opreq2)
have S15: C =
if (C ∈ ℂ , C , **0**) ⟶
(if (A ∈ ℂ , A , **0**) / C) =
(if (A ∈ ℂ , A , **0**) / if (C ∈ ℂ , C , **0**)) **by** (rule MMI_opreq2)
have S16: C =
if (C ∈ ℂ , C , **0**) ⟶
(if (B ∈ ℂ , B , **0**) / C) =

(if (B ∈ ℂ , B , 0) / if (C ∈ ℂ , C , 0)) **by** (rule MMI_opreq2)
from S15 S16 **have** S17: C =
if (C ∈ ℂ , C , 0) ⟶
((if (A ∈ ℂ , A , 0) / C) + (if (B ∈ ℂ , B , 0) / C)) =
((if (A ∈ ℂ , A , 0) / if (C ∈ ℂ , C , 0)) + (if (B ∈ ℂ ,
B , 0) / if (C ∈ ℂ , C , 0))) **by** (rule MMI_opreq12d)
from S14 S17 **have** S18: C =
if (C ∈ ℂ , C , 0) ⟶
(((if (A ∈ ℂ , A , 0) + if (B ∈ ℂ , B , 0)) / C) =
((if (A ∈ ℂ , A , 0) / C) + (if (B ∈ ℂ , B , 0) / C)) ⟷

((if (A ∈ ℂ , A , 0) + if (B ∈ ℂ , B , 0)) / if (C ∈ ℂ , C
, 0)) =
((if (A ∈ ℂ , A , 0) / if (C ∈ ℂ , C , 0)) + (if (B ∈ ℂ ,
B , 0) / if (C ∈ ℂ , C , 0)))) **by** (rule MMI_eqeq12d)
from S13 S18 **have** S19: C =
if (C ∈ ℂ , C , 0) ⟶
((C ≠ 0 ⟶ ((if (A ∈ ℂ , A , 0) + if (B ∈ ℂ , B , 0)) / C
) = ((if (A ∈ ℂ , A , 0) / C) + (if (B ∈ ℂ , B , 0) / C)))
⟷
(if (C ∈ ℂ , C , 0) ≠ 0 ⟶
((if (A ∈ ℂ , A , 0) + if (B ∈ ℂ , B , 0)) / if (C ∈ ℂ , C
, 0)) =
((if (A ∈ ℂ , A , 0) / if (C ∈ ℂ , C , 0)) + (if (B ∈ ℂ ,
B , 0) / if (C ∈ ℂ , C , 0))))) **by** (rule MMI_imbi12d)
have S20: 0 ∈ ℂ **by** (rule MMI_0cn)
from S20 **have** S21: if (A ∈ ℂ , A , 0) ∈ ℂ **by** (rule MMI_elimel)
have S22: 0 ∈ ℂ **by** (rule MMI_0cn)
from S22 **have** S23: if (B ∈ ℂ , B , 0) ∈ ℂ **by** (rule MMI_elimel)
have S24: 0 ∈ ℂ **by** (rule MMI_0cn)
from S24 **have** S25: if (C ∈ ℂ , C , 0) ∈ ℂ **by** (rule MMI_elimel)
from S21 S23 S25 **have** S26: if (C ∈ ℂ , C , 0) ≠ 0 ⟶
((if (A ∈ ℂ , A , 0) + if (B ∈ ℂ , B , 0)) / if (C ∈ ℂ , C
, 0)) =
((if (A ∈ ℂ , A , 0) / if (C ∈ ℂ , C , 0)) + (if (B ∈ ℂ ,
B , 0) / if (C ∈ ℂ , C , 0))) **by** (rule MMI_divdirz)
from S6 S12 S19 S26 **have** S27: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(C ≠ 0 ⟶
((A + B) / C) =
((A / C) + (B / C))) **by** (rule MMI_dedth3h)
from S27 **show** ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ 0) ⟶
((A + B) / C) =
((A / C) + (B / C)) **by** (rule MMI_imp)
qed

lemma (**in** MMIsar0) MMI_divcan3: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ **and**
A3: A ≠ 0
shows ((A · B) / A) = B

```
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: B ∈ ℂ.
   from A1 have S3: A ∈ ℂ.
   from A3 have S4: A ≠ 0.
   from S1 S2 S3 S4 have S5: ( ( A · B ) / A ) = ( A · ( B / A ) ) by
(rule MMI_divass)
   from A1 have S6: A ∈ ℂ.
   from A2 have S7: B ∈ ℂ.
   from A3 have S8: A ≠ 0.
   from S6 S7 S8 have S9: ( A · ( B / A ) ) = B by (rule MMI_divcan2)
   from S5 S9 show ( ( A · B ) / A ) = B by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_divcan4: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ and
    A3: A ≠ 0
   shows ( ( B · A ) / A ) = B
proof -
   from A2 have S1: B ∈ ℂ.
   from A1 have S2: A ∈ ℂ.
   from S1 S2 have S3: ( B · A ) = ( A · B ) by (rule MMI_mulcom)
   from S3 have S4: ( ( B · A ) / A ) = ( ( A · B ) / A ) by (rule MMI_opreq1i)
   from A1 have S5: A ∈ ℂ.
   from A2 have S6: B ∈ ℂ.
   from A3 have S7: A ≠ 0.
   from S5 S6 S7 have S8: ( ( A · B ) / A ) = B by (rule MMI_divcan3)
   from S4 S8 show ( ( B · A ) / A ) = B by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_divcan3z: assumes A1: A ∈ ℂ and
    A2: B ∈ ℂ
   shows A ≠ 0 ⟶ ( ( A · B ) / A ) = B
proof -
   have S1: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 ( A · B ) =
 ( if ( A ≠ 0 , A , 1 ) · B ) by (rule MMI_opreq1)
   have S2: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 A = if ( A ≠ 0 , A , 1 ) by (rule MMI_id)
   from S1 S2 have S3: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 ( ( A · B ) / A ) =
 ( ( if ( A ≠ 0 , A , 1 ) · B ) / if ( A ≠ 0 , A , 1 ) ) by (rule MMI_opreq12d)
   from S3 have S4: A =
 if ( A ≠ 0 , A , 1 ) ⟶
 ( ( ( A · B ) / A ) =
 B ⟷
```

((if (A ≠ **0** , A , **1**) · B) / if (A ≠ **0** , A , **1**)) =
B) **by** (rule MMI_eqeq1d)
  from A1 **have** S5: A ∈ ℂ.
  have S6: **1** ∈ ℂ **by** (rule MMI_1cn)
  from S5 S6 **have** S7: if (A ≠ **0** , A , **1**) ∈ ℂ **by** (rule MMI_keepel)
  from A2 **have** S8: B ∈ ℂ.
  have S9: if (A ≠ **0** , A , **1**) ≠ **0** **by** (rule MMI_elimne0)
  from S7 S8 S9 **have** S10: ((if (A ≠ **0** , A , **1**) · B) / if (A ≠
0 , A , **1**)) =
B **by** (rule MMI_divcan3)
  from S4 S10 **show** A ≠ **0** ⟶ ((A · B) / A) = B **by** (rule MMI_dedth)
qed

lemma (**in** MMIsar0) MMI_divcan4z: **assumes** A1: A ∈ ℂ **and**
   A2: B ∈ ℂ
  shows A ≠ **0** ⟶ ((B · A) / A) = B
proof -
  from A1 **have** S1: A ∈ ℂ.
  from A2 **have** S2: B ∈ ℂ.
  from S1 S2 **have** S3: A ≠ **0** ⟶ ((A · B) / A) = B **by** (rule MMI_divcan3z)
  from A2 **have** S4: B ∈ ℂ.
  from A1 **have** S5: A ∈ ℂ.
  from S4 S5 **have** S6: (B · A) = (A · B) **by** (rule MMI_mulcom)
  from S6 **have** S7: ((B · A) / A) = ((A · B) / A) **by** (rule MMI_opreq1i)
  from S3 S7 **show** A ≠ **0** ⟶ ((B · A) / A) = B **by** (rule MMI_syl5eq)
qed

lemma (**in** MMIsar0) MMI_divcan3t:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((A · B) / A) = B
proof -
  have S1: A =
if (A ∈ ℂ , A , **0**) ⟶
(A ≠ **0** ⟷ if (A ∈ ℂ , A , **0**) ≠ **0**) **by** (rule MMI_neeq1)
  have S2: A =
if (A ∈ ℂ , A , **0**) ⟶
(A · B) =
(if (A ∈ ℂ , A , **0**) · B) **by** (rule MMI_opreq1)
  have S3: A =
if (A ∈ ℂ , A , **0**) ⟶
A = if (A ∈ ℂ , A , **0**) **by** (rule MMI_id)
  from S2 S3 **have** S4: A =
if (A ∈ ℂ , A , **0**) ⟶
((A · B) / A) =
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) **by** (rule MMI_opreq12d)
  from S4 **have** S5: A =
if (A ∈ ℂ , A , **0**) ⟶
(((A · B) / A) =
B ⟷

((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
B) **by** (rule MMI_eqeq1d)
from S1 S5 **have** S6: A =
if (A ∈ ℂ , A , **0**) ⟶
((A ≠ **0** ⟶ ((A · B) / A) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
B)) **by** (rule MMI_imbi12d)
have S7: B =
if (B ∈ ℂ , B , **0**) ⟶
(if (A ∈ ℂ , A , **0**) · B) =
(if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) **by** (rule MMI_opreq2)
from S7 **have** S8: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A ,
0)) **by** (rule MMI_opreq1d)
have S9: B =
if (B ∈ ℂ , B , **0**) ⟶
B = if (B ∈ ℂ , B , **0**) **by** (rule MMI_id)
from S8 S9 **have** S10: B =
if (B ∈ ℂ , B , **0**) ⟶
(((if (A ∈ ℂ , A , **0**) · B) / if (A ∈ ℂ , A , **0**)) =
B ⟷
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A ,
0)) =
if (B ∈ ℂ , B , **0**)) **by** (rule MMI_eqeq12d)
from S10 **have** S11: B =
if (B ∈ ℂ , B , **0**) ⟶
((if (A ∈ ℂ , A , **0**) ≠ **0** ⟶ ((if (A ∈ ℂ , A , **0**) · B) / if
(A ∈ ℂ , A , **0**)) = B) ⟷
(if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A ,
0)) =
if (B ∈ ℂ , B , **0**))) **by** (rule MMI_imbi2d)
have S12: **0** ∈ ℂ **by** (rule MMI_0cn)
from S12 **have** S13: if (A ∈ ℂ , A , **0**) ∈ ℂ **by** (rule MMI_elimel)
have S14: **0** ∈ ℂ **by** (rule MMI_0cn)
from S14 **have** S15: if (B ∈ ℂ , B , **0**) ∈ ℂ **by** (rule MMI_elimel)
from S13 S15 **have** S16: if (A ∈ ℂ , A , **0**) ≠ **0** ⟶
((if (A ∈ ℂ , A , **0**) · if (B ∈ ℂ , B , **0**)) / if (A ∈ ℂ , A ,
0)) =
if (B ∈ ℂ , B , **0**) **by** (rule MMI_divcan3z)
from S6 S11 S16 **have** S17: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A ≠ **0** ⟶ ((A · B) / A) = B) **by** (rule MMI_dedth2h)
from S17 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ A ≠ **0**) ⟶
((A · B) / A) = B **by** (rule MMI_3impia)
qed

lemma (**in** MMIsar0) MMI_divcan4t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
((B $\cdot$ A) / A) = B
proof -
  have S1: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
(A $\cdot$ B) = (B $\cdot$ A) **by** (rule MMI_axmulcom)
  from S1 **have** S2: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
((A $\cdot$ B) / A) = ((B $\cdot$ A) / A) **by** (rule MMI_opreq1d)
  from S2 **have** S3: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
((A $\cdot$ B) / A) = ((B $\cdot$ A) / A) **by** (rule MMI_3adant3)
  have S4: (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
((A $\cdot$ B) / A) = B **by** (rule MMI_divcan3t)
  from S3 S4 **show** (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
((B $\cdot$ A) / A) = B **by** (rule MMI_eqtr3d)
qed

lemma (**in** MMIsar0) MMI_div11: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: B $\in$ $\mathbb{C}$ **and**
   A3: C $\in$ $\mathbb{C}$ **and**
   A4: C $\neq$ **0**
  shows (A / C) = (B / C) $\longleftrightarrow$ A = B
proof -
  from A3 **have** S1: C $\in$ $\mathbb{C}$.
  from A1 **have** S2: A $\in$ $\mathbb{C}$.
  from A3 **have** S3: C $\in$ $\mathbb{C}$.
  from A4 **have** S4: C $\neq$ **0**.
  from S2 S3 S4 **have** S5: (A / C) $\in$ $\mathbb{C}$ **by** (rule MMI_divcl)
  from A2 **have** S6: B $\in$ $\mathbb{C}$.
  from A3 **have** S7: C $\in$ $\mathbb{C}$.
  from A4 **have** S8: C $\neq$ **0**.
  from S6 S7 S8 **have** S9: (B / C) $\in$ $\mathbb{C}$ **by** (rule MMI_divcl)
  from A4 **have** S10: C $\neq$ **0**.
  from S1 S5 S9 S10 **have** S11: (C $\cdot$ (A / C)) =
(C $\cdot$ (B / C)) $\longleftrightarrow$
(A / C) = (B / C) **by** (rule MMI_mulcan)
  from A3 **have** S12: C $\in$ $\mathbb{C}$.
  from A1 **have** S13: A $\in$ $\mathbb{C}$.
  from A4 **have** S14: C $\neq$ **0**.
  from S12 S13 S14 **have** S15: (C $\cdot$ (A / C)) = A **by** (rule MMI_divcan2)
  from A3 **have** S16: C $\in$ $\mathbb{C}$.
  from A2 **have** S17: B $\in$ $\mathbb{C}$.
  from A4 **have** S18: C $\neq$ **0**.
  from S16 S17 S18 **have** S19: (C $\cdot$ (B / C)) = B **by** (rule MMI_divcan2)
  from S15 S19 **have** S20: (C $\cdot$ (A / C)) =
(C $\cdot$ (B / C)) $\longleftrightarrow$ A = B **by** (rule MMI_eqeq12i)
  from S11 S20 **show** (A / C) = (B / C) $\longleftrightarrow$ A = B **by** (rule MMI_bitr3)
qed

lemma (**in** MMIsar0) MMI_div11t:

shows (A ∈ ℂ ∧ B ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (B / C) ⟷ A = B)
proof -
have S1: A =
if (A ∈ ℂ , A , **1**) ⟶
(A / C) =
(if (A ∈ ℂ , A , **1**) / C) **by** (rule MMI_opreq1)
from S1 **have** S2: A =
if (A ∈ ℂ , A , **1**) ⟶
((A / C) =
(B / C) ⟷
(if (A ∈ ℂ , A , **1**) / C) =
(B / C)) **by** (rule MMI_eqeq1d)
have S3: A =
if (A ∈ ℂ , A , **1**) ⟶
(A = B ⟷ if (A ∈ ℂ , A , **1**) = B) **by** (rule MMI_eqeq1)
from S2 S3 **have** S4: A =
if (A ∈ ℂ , A , **1**) ⟶
(((A / C) = (B / C) ⟷ A = B) ⟷
((if (A ∈ ℂ , A , **1**) / C) =
(B / C) ⟷
if (A ∈ ℂ , A , **1**) = B)) **by** (rule MMI_bibi12d)
have S5: B =
if (B ∈ ℂ , B , **1**) ⟶
(B / C) =
(if (B ∈ ℂ , B , **1**) / C) **by** (rule MMI_opreq1)
from S5 **have** S6: B =
if (B ∈ ℂ , B , **1**) ⟶
((if (A ∈ ℂ , A , **1**) / C) =
(B / C) ⟷
(if (A ∈ ℂ , A , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / C)) **by** (rule MMI_eqeq2d)
have S7: B =
if (B ∈ ℂ , B , **1**) ⟶
(if (A ∈ ℂ , A , **1**) =
B ⟷
if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**)) **by** (rule MMI_eqeq2)
from S6 S7 **have** S8: B =
if (B ∈ ℂ , B , **1**) ⟶
(((if (A ∈ ℂ , A , **1**) / C) = (B / C) ⟷ if (A ∈ ℂ , A , **1**
) = B) ⟷
((if (A ∈ ℂ , A , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / C) ⟷
if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**))) **by** (rule MMI_bibi12d)
have S9: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(if (A ∈ ℂ , A , **1**) / C) =

(if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) **by** (rule MMI_opreq2)
have S10: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(if (B ∈ ℂ , B , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) **by** (rule MMI_opreq2)
from S9 S10 **have** S11: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
((if (A ∈ ℂ , A , **1**) / C) =
(if (B ∈ ℂ , B , **1**) / C) ⟷
(if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**))) **by** (rule MMI_eqeq12d)
from S11 **have** S12: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(((if (A ∈ ℂ , A , **1**) / C) = (if (B ∈ ℂ , B , **1**) / C) ⟷ if (A ∈ ℂ , A , **1**) = if (B ∈ ℂ , B , **1**)) ⟷
((if (A ∈ ℂ , A , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) =
(if (B ∈ ℂ , B , **1**) / if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**)) ⟷
if (A ∈ ℂ , A , **1**) =
if (B ∈ ℂ , B , **1**))) **by** (rule MMI_bibi1d)
have S13: **1** ∈ ℂ **by** (rule MMI_1cn)
from S13 **have** S14: if (A ∈ ℂ , A , **1**) ∈ ℂ **by** (rule MMI_elimel)
have S15: **1** ∈ ℂ **by** (rule MMI_1cn)
from S15 **have** S16: if (B ∈ ℂ , B , **1**) ∈ ℂ **by** (rule MMI_elimel)
have S17: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(C ∈ ℂ ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ) **by** (rule MMI_eleq1)
have S18: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(C ≠ **0** ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**) **by** (rule MMI_neeq1)
from S17 S18 **have** S19: C =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
((C ∈ ℂ ∧ C ≠ **0**) ⟷
(if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**)) **by** (rule MMI_anbi12d)
have S20: **1** =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(**1** ∈ ℂ ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ∈ ℂ) **by** (rule MMI_eleq1)
have S21: **1** =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶
(**1** ≠ **0** ⟷
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ≠ **0**) **by** (rule MMI_neeq1)
from S20 S21 **have** S22: **1** =
if ((C ∈ ℂ ∧ C ≠ **0**) , C , **1**) ⟶

(($\mathbf{1}$ ∈ ℂ ∧ $\mathbf{1}$ ≠ $\mathbf{0}$) ⟷
(if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ≠ $\mathbf{0}$)) **by** (rule MMI_anbi12d)
  have S23: $\mathbf{1}$ ∈ ℂ **by** (rule MMI_1cn)
  have S24: $\mathbf{1}$ ≠ $\mathbf{0}$ **by** (rule MMI_ax1ne0)
  from S23 S24 **have** S25: $\mathbf{1}$ ∈ ℂ ∧ $\mathbf{1}$ ≠ $\mathbf{0}$ **by** (rule MMI_pm3_2i)
  from S19 S22 S25 **have** S26: if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ≠ $\mathbf{0}$ **by** (rule MMI_elimhyp)
  from S26 **have** S27: if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ∈ ℂ **by** (rule MMI_pm3_26i)
  from S26 **have** S28: if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ∈ ℂ ∧ if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ≠ $\mathbf{0}$ **.**
  from S28 **have** S29: if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$) ≠ $\mathbf{0}$ **by** (rule MMI_pm3_27i)
  from S14 S16 S27 S29 **have** S30: (if (A ∈ ℂ , A , $\mathbf{1}$) / if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$)) =
(if (B ∈ ℂ , B , $\mathbf{1}$) / if ((C ∈ ℂ ∧ C ≠ $\mathbf{0}$) , C , $\mathbf{1}$)) ⟷
if (A ∈ ℂ , A , $\mathbf{1}$) =
if (B ∈ ℂ , B , $\mathbf{1}$) **by** (rule MMI_div11)
  from S4 S8 S12 S30 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ $\mathbf{0}$)) ⟶

((A / C) = (B / C) ⟷ A = B) **by** (rule MMI_dedth3h)
qed

end

105 Metamath examples

theory MMI_examples **imports** MMI_Complex_ZF

begin

This theory contains 10 theorems translated from Metamath (with proofs). It is included in the proof document as an illustration of how a translated Metamath proof looks like. The "known_theorems.txt" file included in the IsarMathLib distribution provides a list of all translated facts.

lemma (**in** MMIsar0) MMI_dividt:
  shows (A ∈ ℂ ∧ A ≠ $\mathbf{0}$) ⟶ (A / A) = $\mathbf{1}$
proof -
  have S1: (A ∈ ℂ ∧ A ∈ ℂ ∧ A ≠ $\mathbf{0}$) ⟶
(A / A) = (A · ($\mathbf{1}$ / A)) **by** (rule MMI_divrect)
  from S1 **have** S2: ((A ∈ ℂ ∧ A ∈ ℂ) ∧ A ≠ $\mathbf{0}$) ⟶
(A / A) = (A · ($\mathbf{1}$ / A)) **by** (rule MMI_3expa)
  from S2 **have** S3: (A ∈ ℂ ∧ A ≠ $\mathbf{0}$) ⟶
(A / A) = (A · ($\mathbf{1}$ / A)) **by** (rule MMI_anabsan)
  have S4: (A ∈ ℂ ∧ A ≠ $\mathbf{0}$) ⟶
(A · ($\mathbf{1}$ / A)) = $\mathbf{1}$ **by** (rule MMI_recidt)

from S3 S4 show (A ∈ ℂ ∧ A ≠ **0**) ⟶ (A / A) = **1** by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_div0t:
shows (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**0** / A) = **0**
proof -
have S1: **0** ∈ ℂ by (rule MMI_0cn)
have S2: (**0** ∈ ℂ ∧ A ∈ ℂ ∧ A ≠ **0**) ⟶
(**0** / A) = (**0** · (**1** / A)) by (rule MMI_divrect)
from S1 S2 have S3: (A ∈ ℂ ∧ A ≠ **0**) ⟶
(**0** / A) = (**0** · (**1** / A)) by (rule MMI_mp3an1)
have S4: (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**1** / A) ∈ ℂ by (rule MMI_recclt)
have S5: (**1** / A) ∈ ℂ ⟶ (**0** · (**1** / A)) = **0**
by (rule MMI_mul02t)
from S4 S5 have S6: (A ∈ ℂ ∧ A ≠ **0**) ⟶
(**0** · (**1** / A)) = **0** by (rule MMI_syl)
from S3 S6 show (A ∈ ℂ ∧ A ≠ **0**) ⟶ (**0** / A) = **0** by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_diveq0t:
shows (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) = **0** ⟷ A = **0**)
proof -
have S1: (C ∈ ℂ ∧ C ≠ **0**) ⟶ (**0** / C) = **0** by (rule MMI_div0t)
from S1 have S2: (C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) =
(**0** / C) ⟷ (A / C) = **0**) by (rule MMI_eqeq2d)
from S2 have S3: (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) =
(**0** / C) ⟷ (A / C) = **0**) by (rule MMI_3adant1)
have S4: **0** ∈ ℂ by (rule MMI_0cn)
have S5: (A ∈ ℂ ∧ **0** ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (**0** / C) ⟷ A = **0**) by (rule MMI_div11t)
from S4 S5 have S6: (A ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (**0** / C) ⟷ A = **0**) by (rule MMI_mp3an2)
from S6 have S7: (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) = (**0** / C) ⟷ A = **0**) by (rule MMI_3impb)
from S3 S7 show (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) = **0** ⟷ A = **0**) by (rule MMI_bitr3d)
qed

lemma (in MMIsar0) MMI_recrec: assumes A1: A ∈ ℂ and
A2: A ≠ **0**
shows (**1** / (**1** / A)) = A
proof -
from A1 have S1: A ∈ ℂ.
from A2 have S2: A ≠ **0**.
from S1 S2 have S3: (**1** / A) ∈ ℂ by (rule MMI_reccl)
have S4: **1** ∈ ℂ by (rule MMI_1cn)

```
   from A1 have S5: A ∈ ℂ.
   have S6: 1 ≠ 0 by (rule MMI_ax1ne0)
   from A2 have S7: A ≠ 0.
   from S4 S5 S6 S7 have S8: ( 1 / A ) ≠ 0 by (rule MMI_divne0)
   from S3 S8 have S9: ( ( 1 / A ) · ( 1 / ( 1 / A ) ) ) = 1
     by (rule MMI_recid)
   from S9 have S10: ( A · ( ( 1 / A ) · ( 1 / ( 1 / A ) ) ) ) =
 ( A · 1 ) by (rule MMI_opreq2i)
   from A1 have S11: A ∈ ℂ.
   from A2 have S12: A ≠ 0.
   from S11 S12 have S13: ( A · ( 1 / A ) ) = 1 by (rule MMI_recid)
   from S13 have S14: ( ( A · ( 1 / A ) ) · ( 1 / ( 1 / A ) ) ) =
 ( 1 · ( 1 / ( 1 / A ) ) ) by (rule MMI_opreq1i)
   from A1 have S15: A ∈ ℂ.
   from S3 have S16: ( 1 / A ) ∈ ℂ .
   from S3 have S17: ( 1 / A ) ∈ ℂ .
   from S8 have S18: ( 1 / A ) ≠ 0 .
   from S17 S18 have S19: ( 1 / ( 1 / A ) ) ∈ ℂ by (rule MMI_reccl)
   from S15 S16 S19 have S20:
     ( ( A · ( 1 / A ) ) · ( 1 / ( 1 / A ) ) ) =
 ( A · ( ( 1 / A ) · ( 1 / ( 1 / A ) ) ) ) by (rule MMI_mulass)
   from S19 have S21: ( 1 / ( 1 / A ) ) ∈ ℂ .
   from S21 have S22: ( 1 · ( 1 / ( 1 / A ) ) ) =
 ( 1 / ( 1 / A ) ) by (rule MMI_mulid2)
   from S14 S20 S22 have S23:
     ( A · ( ( 1 / A ) · ( 1 / ( 1 / A ) ) ) ) =
 ( 1 / ( 1 / A ) ) by (rule MMI_3eqtr3)
   from A1 have S24: A ∈ ℂ.
   from S24 have S25: ( A · 1 ) = A by (rule MMI_mulid1)
   from S10 S23 S25 show ( 1 / ( 1 / A ) ) = A by (rule MMI_3eqtr3)
qed

lemma (in MMIsar0) MMI_divid: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ( A / A ) = 1
proof -
   from A1 have S1: A ∈ ℂ.
   from A1 have S2: A ∈ ℂ.
   from A2 have S3: A ≠ 0.
   from S1 S2 S3 have S4: ( A / A ) = ( A · ( 1 / A ) ) by (rule MMI_divrec)
   from A1 have S5: A ∈ ℂ.
   from A2 have S6: A ≠ 0.
   from S5 S6 have S7: ( A · ( 1 / A ) ) = 1 by (rule MMI_recid)
   from S4 S7 show ( A / A ) = 1 by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_div0: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ( 0 / A ) = 0
```

```
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: A ≠ 0.
   have S3: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 0 / A ) = 0 by (rule MMI_div0t)
   from S1 S2 S3 show ( 0 / A ) = 0 by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_div1: assumes A1: A ∈ ℂ
   shows ( A / 1 ) = A
proof -
   from A1 have S1: A ∈ ℂ.
   from S1 have S2: ( 1 · A ) = A by (rule MMI_mulid2)
   from A1 have S3: A ∈ ℂ.
   have S4: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S5: A ∈ ℂ.
   have S6: 1 ≠ 0 by (rule MMI_ax1ne0)
   from S3 S4 S5 S6 have S7: ( A / 1 ) = A ⟷ ( 1 · A ) = A
     by (rule MMI_divmul)
   from S2 S7 show ( A / 1 ) = A by (rule MMI_mpbir)
qed

lemma (in MMIsar0) MMI_div1t:
   shows A ∈ ℂ ⟶ ( A / 1 ) = A
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( A / 1 ) =
 ( if ( A ∈ ℂ , A , 1 ) / 1 ) by (rule MMI_opreq1)
   have S2: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 A = if ( A ∈ ℂ , A , 1 ) by (rule MMI_id)
   from S1 S2 have S3: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( ( A / 1 ) =
 A ⟷
 ( if ( A ∈ ℂ , A , 1 ) / 1 ) =
 if ( A ∈ ℂ , A , 1 ) ) by (rule MMI_eqeq12d)
   have S4: 1 ∈ ℂ by (rule MMI_1cn)
   from S4 have S5: if ( A ∈ ℂ , A , 1 ) ∈ ℂ by (rule MMI_elimel)
   from S5 have S6: ( if ( A ∈ ℂ , A , 1 ) / 1 ) =
 if ( A ∈ ℂ , A , 1 ) by (rule MMI_div1)
   from S3 S6 show A ∈ ℂ ⟶ ( A / 1 ) = A by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_divnegt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( - ( A / B ) ) = ( ( - A ) / B )
proof -
   have S1: ( A ∈ ℂ ∧ ( 1 / B ) ∈ ℂ ) ⟶
```

((- A) · (**1** / B)) =
(- (A · (**1** / B))) **by** (rule MMI_mulneg1t)
  have S2: (B ∈ ℂ ∧ B ≠ **0**) ⟶ (**1** / B) ∈ ℂ **by** (rule MMI_recclt)
  from S1 S2 **have** S3: (A ∈ ℂ ∧ (B ∈ ℂ ∧ B ≠ **0**)) ⟶
((- A) · (**1** / B)) =
(- (A · (**1** / B))) **by** (rule MMI_sylan2)
  from S3 **have** S4: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
((- A) · (**1** / B)) =
(- (A · (**1** / B))) **by** (rule MMI_3impb)
  have S5: ((- A) ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
((- A) / B) =
((- A) · (**1** / B)) **by** (rule MMI_divrect)
  have S6: A ∈ ℂ ⟶ (- A) ∈ ℂ **by** (rule MMI_negclt)
  from S5 S6 **have** S7: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
((- A) / B) =
((- A) · (**1** / B)) **by** (rule MMI_syl3an1)
  have S8: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(A / B) = (A · (**1** / B)) **by** (rule MMI_divrect)
  from S8 **have** S9: (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(- (A / B)) =
(- (A · (**1** / B))) **by** (rule MMI_negeqd)
  from S4 S7 S9 **show** (A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ **0**) ⟶
(- (A / B)) = ((- A) / B) **by** (rule MMI_3eqtr4rd)
qed

lemma (**in** MMIsar0) MMI_divsubdirt:
  shows ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A - B) / C) =
((A / C) - (B / C))
proof -
  have S1: ((A ∈ ℂ ∧ (- B) ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + (- B)) / C) =
((A / C) + ((- B) / C)) **by** (rule MMI_divdirt)
  have S2: B ∈ ℂ ⟶ (- B) ∈ ℂ **by** (rule MMI_negclt)
  from S1 S2 **have** S3: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + (- B)) / C) =
((A / C) + ((- B) / C)) **by** (rule MMI_syl3anl2)
  have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + (- B)) = (A - B) **by** (rule MMI_negsubt)
  from S4 **have** S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- B)) = (A - B) **by** (rule MMI_3adant3)
  from S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + (- B)) / C) =
((A - B) / C) **by** (rule MMI_opreq1d)
  from S6 **have** S7: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + (- B)) / C) =
((A - B) / C) **by** (rule MMI_adantr)
  have S8: (B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(- (B / C)) = ((- B) / C) **by** (rule MMI_divnegt)

```
  from S8 have S9: ( ( B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( - ( B / C ) ) = ( ( - B ) / C ) by (rule MMI_3expa)
  from S9 have S10: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( - ( B / C ) ) = ( ( - B ) / C ) by (rule MMI_3adantl1)
  from S10 have S11: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( ( A / C ) + ( - ( B / C ) ) ) =
 ( ( A / C ) + ( ( - B ) / C ) ) by (rule MMI_opreq2d)
  have S12: ( ( A / C ) ∈ ℂ ∧ ( B / C ) ∈ ℂ ) ⟶
 ( ( A / C ) + ( - ( B / C ) ) ) =
 ( ( A / C ) - ( B / C ) ) by (rule MMI_negsubt)
  have S13: ( A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ 0 ) ⟶
 ( A / C ) ∈ ℂ by (rule MMI_divclt)
  from S13 have S14: ( ( A ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( A / C ) ∈ ℂ by (rule MMI_3expa)
  from S14 have S15: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( A / C ) ∈ ℂ by (rule MMI_3adantl2)
  have S16: ( B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ 0 ) ⟶
 ( B / C ) ∈ ℂ by (rule MMI_divclt)
  from S16 have S17: ( ( B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( B / C ) ∈ ℂ by (rule MMI_3expa)
  from S17 have S18: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( B / C ) ∈ ℂ by (rule MMI_3adantl1)
  from S12 S15 S18 have S19: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0
) ⟶
 ( ( A / C ) + ( - ( B / C ) ) ) =
 ( ( A / C ) - ( B / C ) ) by (rule MMI_sylanc)
  from S11 S19 have S20: ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶

 ( ( A / C ) + ( ( - B ) / C ) ) =
 ( ( A / C ) - ( B / C ) ) by (rule MMI_eqtr3d)
  from S3 S7 S20 show ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( ( A - B ) / C ) =
 ( ( A / C ) - ( B / C ) ) by (rule MMI_3eqtr3d)
qed


end
```

106 Metamath interface

theory Metamath_Interface **imports** Complex_ZF MMI_prelude

begin

This theory contains some lemmas that make it possible to use the theorems translated from Metamath in a the `complex0` context.

106.1 MMisar0 and complex0 contexts.

In the section we show a lemma that the assumptions in `complex0` context imply the assumptions of the `MMIsar0` context. The `Metamath_sampler` theory provides examples how this lemma can be used.

The next lemma states that we can use the theorems proven in the `MMIsar0` context in the `complex0` context. Unfortunately we have to use low level Isabelle methods "rule" and "unfold" in the proof, simp and blast fail on the order axioms.

```
lemma (in complex0) MMIsar_valid:
  shows MMIsar0(ℝ,ℂ,1,0,i,CplxAdd(R,A),CplxMul(R,A,M),
  StrictVersion(CplxROrder(R,A,r)))
proof -
  let real = ℝ
  let complex = ℂ
  let zero = 0
  let one = 1
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have (∀a b. a ∈ real ∧ b ∈ real ⟶
    ⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel))
  proof -
    have I:
      ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ (a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a))
      using pre_axlttri by blast
    { fix a b assume a ∈ real ∧ b ∈ real
      with I have (a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a))
 by blast
      hence
 ⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)
 by simp
    } thus (∀a b. a ∈ real ∧ b ∈ real ⟶
 (⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)))
      by blast
  qed
  moreover
  have (∀a b c.
    a ∈ real ∧ b ∈ real ∧ c ∈ real ⟶
    ⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel)
  proof -
    have II: ∀a b c.  a ∈ ℝ ∧ b ∈ ℝ ∧ c ∈ ℝ ⟶
      ((a <ℝ b ∧ b <ℝ c) ⟶ a <ℝ c)
      using pre_axlttrn by blast
    { fix a b c assume a ∈ real ∧ b ∈ real ∧ c ∈ real
      with II have (a <ℝ b ∧ b <ℝ c) ⟶ a <ℝ c
 by blast
```

```
     hence
⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel
by simp
   } thus  (∀a b c.
a ∈ real ∧ b ∈ real ∧ c ∈ real ⟶
⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel)
     by blast
 qed
 moreover have (∀A B C.
   A ∈ real ∧ B ∈ real ∧ C ∈ real ⟶
   ⟨A, B⟩ ∈ lessrrel ⟶
   ⟨caddset  ⟨C, A⟩, caddset  ⟨C, B⟩⟩ ∈ lessrrel)
   using pre_axltadd by simp
 moreover have (∀A B. A ∈ real ∧ B ∈ real ⟶
   ⟨zero, A⟩ ∈ lessrrel ∧ ⟨zero, B⟩ ∈ lessrrel ⟶
   ⟨zero, cmulset  ⟨A, B⟩⟩ ∈ lessrrel)
   using pre_axmulgt0 by simp
 moreover have
   (∀S. S ⊆ real ∧ S ≠ 0 ∧ (∃x∈real. ∀y∈S. ⟨y, x⟩ ∈ lessrrel) ⟶
   (∃x∈real.
   (∀y∈S. ⟨x, y⟩ ∉ lessrrel) ∧
   (∀y∈real. ⟨y, x⟩ ∈ lessrrel ⟶ (∃z∈S. ⟨y, z⟩ ∈ lessrrel))))
   using pre_axsup by simp
 moreover have ℝ ⊆ ℂ using axresscn by simp
 moreover have 1 ≠ 0 using ax1ne0 by simp
 moreover have ℂ isASet by simp
 moreover have  CplxAdd(R,A) : ℂ × ℂ → ℂ
   using axaddopr by simp
 moreover have CplxMul(R,A,M) : ℂ × ℂ → ℂ
   using axmulopr by simp
 moreover have
   ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a· b = b · a
   using axmulcom by simp
 hence (∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
         cmulset  ⟨a, b⟩ = cmulset  ⟨b, a⟩
   ) by simp
 moreover have ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a + b ∈ ℂ
   using axaddcl by simp
 hence (∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
         caddset  ⟨a, b⟩ ∈ ℂ
     ) by simp
 moreover have ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a · b ∈ ℂ
   using axmulcl by simp
 hence (∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
   cmulset  ⟨a, b⟩ ∈ ℂ ) by simp
 moreover have
   ∀a b C. a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
   a · (b + C) = a · b + a · C
   using axdistr by simp
```

```
 hence ∀a b C.
        a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
        cmulset ⟨a, caddset ⟨b, C⟩⟩ =
        caddset
        ⟨cmulset ⟨a, b⟩, cmulset ⟨a, C⟩⟩
   by simp
 moreover have ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
        a + b = b + a
   using axaddcom by simp
 hence ∀a b.
         a ∈ ℂ ∧ b ∈ ℂ ⟶
         caddset ⟨a, b⟩ = caddset ⟨b, a⟩ by simp
 moreover have ∀a b C. a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
     a + b + C = a + (b + C)
   using axaddass by simp
 hence ∀a b C.
         a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
         caddset ⟨caddset ⟨a, b⟩, C⟩ =
         caddset ⟨a, caddset ⟨b, C⟩⟩ by simp
 moreover have
   ∀a b c. a ∈ ℂ ∧ b ∈ ℂ ∧ c ∈ ℂ ⟶ a·b·c = a·(b·c)
   using axmulass by simp
 hence ∀a b C.
         a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
         cmulset ⟨cmulset ⟨a, b⟩, C⟩ =
         cmulset ⟨a, cmulset ⟨b, C⟩⟩ by simp
 moreover have 1 ∈ ℝ using ax1re by simp
 moreover have i·i + 1 = 0
   using axi2m1 by simp
 hence caddset ⟨cmulset ⟨i, i⟩, 1⟩ = 0 by simp
 moreover have ∀a. a ∈ ℂ ⟶ a + 0 = a
   using ax0id by simp
 hence ∀a. a ∈ ℂ ⟶ caddset ⟨a, 0⟩ = a by simp
 moreover have i ∈ ℂ using axicn by simp
 moreover have ∀a. a ∈ ℂ ⟶ (∃x∈ℂ. a + x = 0)
   using axnegex by simp
 hence ∀a. a ∈ ℂ ⟶
   (∃x∈ℂ. caddset ⟨a, x⟩ = 0) by simp
 moreover have ∀a. a ∈ ℂ ∧ a ≠ 0 ⟶ (∃x∈ℂ. a · x = 1)
   using axrecex by simp
 hence ∀a. a ∈ ℂ ∧ a ≠ 0 ⟶
     ( ∃x∈ℂ. cmulset ⟨a, x⟩ = 1 ) by simp
 moreover have ∀a. a ∈ ℂ ⟶ a·1 = a
   using ax1id by simp
hence  ∀a. a ∈ ℂ ⟶
       cmulset ⟨a, 1⟩ = a by simp
moreover have ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ a + b ∈ ℝ
  using axaddrcl by simp
hence ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶
```

```
    caddset ⟨a, b⟩ ∈ ℝ by simp
moreover have ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ a · b ∈ ℝ
  using axmulrcl by simp
hence ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶
    cmulset ⟨a, b⟩ ∈ ℝ by simp
moreover have ∀a. a ∈ ℝ ⟶ (∃x∈ℝ. a + x = 0)
  using axrnegex by simp
hence ∀a. a ∈ ℝ ⟶
  ( ∃x∈ℝ. caddset ⟨a, x⟩ = 0 ) by simp
moreover have ∀a. a ∈ ℝ ∧ a≠0 ⟶ (∃x∈ℝ. a · x = 1)
  using axrrecex by simp
hence ∀a. a ∈ ℝ ∧ a ≠ 0 ⟶
  ( ∃x∈ℝ. cmulset ⟨a, x⟩ = 1) by simp

 ultimately have
(
  (
    (
      ( ∀a b.
        a ∈ ℝ ∧ b ∈ ℝ ⟶
        ⟨a, b⟩ ∈ lessrrel ⟷
        ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)
      ) ∧

      ( ∀a b C.
        a ∈ ℝ ∧ b ∈ ℝ ∧ C ∈ ℝ ⟶
        ⟨a, b⟩ ∈ lessrrel ∧
        ⟨b, C⟩ ∈ lessrrel ⟶
        ⟨a, C⟩ ∈ lessrrel
      ) ∧

      (∀a b C.
        a ∈ ℝ ∧ b ∈ ℝ ∧ C ∈ ℝ ⟶
        ⟨a, b⟩ ∈ lessrrel ⟶
        ⟨caddset ⟨C, a⟩, caddset ⟨C, b⟩⟩ ∈
        lessrrel
      )
    ) ∧

    (
      ( ∀a b.
        a ∈ ℝ ∧ b ∈ ℝ ⟶
        ⟨0, a⟩ ∈ lessrrel ∧
        ⟨0, b⟩ ∈ lessrrel ⟶
        ⟨0, cmulset ⟨a, b⟩⟩ ∈
        lessrrel
      ) ∧

      ( ∀S. S ⊆ ℝ ∧ S ≠ 0 ∧
```

```
          ( ∃x∈ℝ. ∀y∈S. ⟨y, x⟩ ∈ lessrrel
          ) ⟶
          ( ∃x∈ℝ.
             ( ∀y∈S. ⟨x, y⟩ ∉ lessrrel
             ) ∧
             ( ∀y∈ℝ. ⟨y, x⟩ ∈ lessrrel ⟶
                ( ∃z∈S. ⟨y, z⟩ ∈ lessrrel
                )
             )
          )
       )
    ) ∧

    ℝ ⊆ ℂ ∧
    1 ≠ 0
 ) ∧

 ( ℂ isASet ∧ caddset ∈ ℂ × ℂ → ℂ ∧
  cmulset ∈ ℂ × ℂ → ℂ
 ) ∧

 (
    (∀a b.
        a ∈ ℂ ∧ b ∈ ℂ ⟶
        cmulset  ⟨a, b⟩ = cmulset  ⟨b, a⟩
    ) ∧

    (∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
        caddset  ⟨a, b⟩ ∈ ℂ
    )

 ) ∧

 (∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
    cmulset  ⟨a, b⟩ ∈ ℂ
 ) ∧

 (∀a b C.
       a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
       cmulset  ⟨a, caddset  ⟨b, C⟩⟩ =
       caddset
       ⟨cmulset  ⟨a, b⟩, cmulset  ⟨a, C⟩⟩
 )
) ∧


(
 (
    (∀a b.
```

```
         a ∈ ℂ ∧ b ∈ ℂ ⟶
         caddset ⟨a, b⟩ = caddset ⟨b, a⟩
      ) ∧

      (∀a b C.
         a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
         caddset ⟨caddset ⟨a, b⟩, C⟩ =
         caddset ⟨a, caddset ⟨b, C⟩⟩
      ) ∧

      (∀a b C.
         a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
         cmulset ⟨cmulset ⟨a, b⟩, C⟩ =
         cmulset ⟨a, cmulset ⟨b, C⟩⟩
      )
   ) ∧


   (1 ∈ ℝ ∧
    caddset ⟨cmulset ⟨i, i⟩, 1⟩ = 0
   ) ∧

   (∀a. a ∈ ℂ ⟶ caddset ⟨a, 0⟩ = a
   ) ∧

   i ∈ ℂ
) ∧

(
   (∀a. a ∈ ℂ ⟶
      (∃x∈ℂ. caddset ⟨a, x⟩ = 0
      )
   ) ∧

   ( ∀a. a ∈ ℂ ∧ a ≠ 0 ⟶
      ( ∃x∈ℂ. cmulset ⟨a, x⟩ = 1
      )
   ) ∧

   ( ∀a. a ∈ ℂ ⟶
        cmulset ⟨a, 1⟩ = a
   )
) ∧

(
   ( ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶
     caddset ⟨a, b⟩ ∈ ℝ
   ) ∧
```

```
    ( ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶
      cmulset ⟨a, b⟩ ∈ ℝ
    )
) ∧

( ∀a. a ∈ ℝ ⟶
    ( ∃x∈ℝ. caddset ⟨a, x⟩ = 0
    )
) ∧

( ∀a. a ∈ ℝ ∧ a ≠ 0 ⟶
    ( ∃x∈ℝ. cmulset ⟨a, x⟩ = 1
    )
)
  by blast
then show MMIsar0(ℝ,ℂ,1,0,i,CplxAdd(R,A),CplxMul(R,A,M),
  StrictVersion(CplxROrder(R,A,r))) unfolding MMIsar0_def by blast
qed


end
```

107 Metamath sampler

theory Metamath_Sampler **imports** Metamath_Interface MMI_Complex_ZF_2

begin

The theorems translated from Metamath reside in the MMI_Complex_ZF, MMI_Complex_ZF_1 and MMI_Complex_ZF_2 theories. The proofs of these theorems are very verbose and for this reason the theories are not shown in the proof document or the FormaMath.org site. This theory file contains some examples of theorems translated from Metamath and formulated in the complex0 context. This serves two purposes: to give an overview of the material covered in the translated theorems and to provide examples of how to take a translated theorem (proven in the MMIsar0 context) and transfer it to the complex0 context. The typical procedure for moving a theorem from MMIsar0 to complex0 is as follows: First we define certain aliases that map names defined in the complex0 to their corresponding names in the MMIsar0 context. This makes it easy to copy and paste the statement of the theorem as displayed with ProofGeneral. Then we run the Isabelle from ProofGeneral up to the theorem we want to move. When the theorem is verified ProofGeneral displays the statement in the raw set theory notation, stripped from any notation defined in the MMIsar0 locale. This is what we copy to the proof in the complex0 locale. After that we just can write "then have ?thesis by simp" and the simplifier translates the raw set theory notation to the one used in complex0.

107.1 Extended reals and order

In this section we import a couple of theorems about the extended real line and the linear order on it.

Metamath uses the set of real numbers extended with $+\infty$ and $-\infty$. The $+\infty$ and $-\infty$ symbols are defined quite arbitrarily as $\mathbb{C}$ and $\{\mathbb{C}\}$, respectively. The next lemma that corresponds to Metamath's `renfdisj` states that $+\infty$ and $-\infty$ are not elements of $\mathbb{R}$.

```
lemma (in complex0) renfdisj: shows ℝ ∩ {+∞,−∞} = 0
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have real ∩ {complex, {complex}} = 0
    by (rule MMIsar0.MMI_renfdisj)
  thus ℝ ∩ {+∞,−∞} = 0 by simp
qed
```

The order relation used most often in Metamath is defined on the set of complex reals extended with $+\infty$ and $-\infty$. The next lemma allows to use Metamath's `xrltso` that states that the `<` relations is a strict linear order on the extended set.

```
lemma (in complex0) xrltso: shows < Orders ℝ*
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    (lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
      {{complex}} × real) Orders (real ∪ {complex, {complex}})
```

```
    by (rule MMIsar0.MMI_xrltso)
  moreover have lessrrel ∩ real × real = lessrrel
    using cplx_strict_ord_on_cplx_reals by auto
  ultimately show < Orders ℝ* by simp
qed
```

Metamath defines the usual $<$ and $\leq$ ordering relations for the extended real line, including $+\infty$ and $-\infty$.

```
lemma (in complex0) xrrebndt: assumes A1: x ∈ ℝ*
  shows x ∈ ℝ ⟷ ( −∞ < x ∧ x < +∞ )
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have x ∈ ℝ ∪ {ℂ, {ℂ}} ⟶
    x ∈ ℝ ⟷ ⟨{ℂ}, x⟩ ∈ lessrrel ∩ ℝ × ℝ ∪ {⟨{ℂ}, ℂ⟩} ∪
    ℝ × {ℂ} ∪ {{ℂ}} × ℝ ∧
    ⟨x, ℂ⟩ ∈ lessrrel ∩ ℝ × ℝ ∪ {⟨{ℂ}, ℂ⟩} ∪
    ℝ × {ℂ} ∪ {{ℂ}} × ℝ
    by (rule MMIsar0.MMI_xrrebndt)
  then have x ∈ ℝ* ⟶ ( x ∈ ℝ ⟷ ( −∞ < x ∧ x < +∞ ) )
    by simp
  with A1 show thesis by simp
qed
```

A quite involved inequality.

```
lemma (in complex0) lt2mul2divt:
  assumes A1: a ∈ ℝ  b ∈ ℝ  c ∈ ℝ  d ∈ ℝ and
  A2: 0 < b  0 < d
  shows a·b < c·d ⟷ a/d < c/b
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
```

```
    using MMIsar_valid by simp
  then have
    (a ∈ real ∧ b ∈ real) ∧
    (c ∈ real ∧ d ∈ real) ∧
    ⟨zero, b⟩ ∈ lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪ {{complex}} × real ∧
    ⟨zero, d⟩ ∈ lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪ {{complex}} × real ⟶
    ⟨cmulset ⟨a, b⟩, cmulset ⟨c, d⟩⟩ ∈
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real ⟷
    ⟨⋃{x ∈ complex . cmulset ⟨d, x⟩ = a},
    ⋃{x ∈ complex . cmulset ⟨b, x⟩ = c}⟩ ∈
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real
    by (rule MMIsar0.MMI_lt2mul2divt)
  with A1 A2 show thesis by simp
qed
```

A real number is smaller than its half iff it is positive.

```
lemma (in complex0) halfpos: assumes A1: a ∈ ℝ
  shows 0 < a ⟷ a/2 < a
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  from A1 have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    and a ∈ real
    using MMIsar_valid by auto
  then have
    ⟨zero, a⟩ ∈
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real ⟷
    ⟨⋃{x ∈ complex . cmulset ⟨caddset ⟨one, one⟩, x⟩ = a}, a⟩ ∈
    lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪ {{complex}} × real
    by (rule MMIsar0.MMI_halfpos)
  then show thesis by simp
qed
```

One more inequality.

```
lemma (in complex0) ledivp1t:
  assumes A1: a ∈ ℝ  b ∈ ℝ and
```

```
  A2: 0 ≤ a  0 ≤ b
  shows (a/(b + 1))·b ≤ a
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    (a ∈ real ∧ ⟨a, zero⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real) ∧
    b ∈ real ∧ ⟨b, zero⟩ ∉ lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
    {{complex}} × real ⟶
    ⟨a,cmulset⟨⋃{x ∈ complex . cmulset⟨caddset⟨b, one⟩, x⟩ = a}, b⟩⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real
    by (rule MMIsar0.MMI_ledivp1t)
  with A1 A2 show thesis by simp
qed
```

107.2 Natural real numbers

In standard mathematics natural numbers are treated as a subset of real numbers. From the set theory point of view however those are quite different objects. In this section we talk about "real natural" numbers i.e. the conterpart of natural numbers that is a subset of the reals.

Two ways of saying that there are no natural numbers between n and $n+1$.

```
lemma (in complex0) no_nats_between:
  assumes A1: n ∈ ℕ  k ∈ ℕ
  shows
  n≤k ⟷ n < k+1
  n < k ⟷ n + 1 ≤ k
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
```

```
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    n ∈ ⋂{N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset  ⟨n, one⟩ ∈ N)} ∧
    k ∈ ⋂{N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset  ⟨n, one⟩ ∈ N)} ⟶
    ⟨k, n⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪ real × {complex}
∪
    {{complex}} × real ⟷
    ⟨n, caddset  ⟨k, one⟩⟩ ∈
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪ real × {complex}
∪
    {{complex}} × real by (rule MMIsar0.MMI_nnleltp1t)
  then have n ∈ ℕ ∧ k ∈ ℕ ⟶ n ≤ k ⟷ n < k + 1
    by simp
  with A1 show n≤k ⟷ n < k+1 by simp
  from I have
    n ∈ ⋂{N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset  ⟨n, one⟩ ∈ N)} ∧
    k ∈ ⋂{N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset  ⟨n, one⟩ ∈ N)} ⟶
    ⟨n, k⟩ ∈
    lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
    {{complex}} × real ⟷  ⟨k, caddset  ⟨n, one⟩⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪ real × {complex}
∪
    {{complex}} × real by (rule MMIsar0.MMI_nnltp1let)
  then have n ∈ ℕ ∧ k ∈ ℕ ⟶  n < k ⟷ n + 1 ≤ k
    by simp
  with A1 show n < k ⟷ n + 1 ≤ k by simp
qed
```

Metamath has some very complicated and general version of induction on (complex) natural numbers that I can't even understand. As an exercise I derived a more standard version that is imported to the `complex0` context below.

```
lemma (in complex0) cplx_nat_ind: assumes A1: ψ(1) and
  A2: ∀k ∈ ℕ. ψ(k) ⟶ ψ(k+1) and
  A3: n ∈ ℕ
  shows ψ(n)
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
```

```
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  moreover from A1 A2 A3 have
    ψ(one)
    ∀k∈⋂{N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset ⟨n, one⟩ ∈ N)}.
    ψ(k) ⟶ ψ(caddset ⟨k, one⟩)
    n ∈ ⋂{N ∈ Pow(real) . one ∈ N ∧
    (∀n. n ∈ N ⟶ caddset ⟨n, one⟩ ∈ N)}
    by auto
  ultimately show ψ(n) by (rule MMIsar0.nnind1)
qed
```

Some simple arithmetics.

```
lemma (in complex0) arith: shows
  2 + 2 = 4
  2·2 = 4
  3·2 = 6
  3·3 = 9
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    caddset ⟨caddset ⟨one, one⟩, caddset ⟨one, one⟩⟩ =
    caddset ⟨caddset ⟨caddset ⟨one, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI_2p2e4)
  thus 2 + 2 = 4 by simp
  from I have
    cmulset⟨caddset⟨one, one⟩, caddset⟨one, one⟩⟩ =
    caddset⟨caddset⟨caddset⟨one, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI_2t2e4)
  thus 2·2 = 4 by simp
  from I have
    cmulset⟨caddset⟨caddset⟨one, one⟩, one⟩, caddset⟨one, one⟩⟩ =
```

```
    caddset ⟨caddset⟨caddset⟨caddset⟨caddset
    ⟨one, one⟩, one⟩, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI_3t2e6)
  thus 3·2 = 6 by simp
  from I have cmulset
    ⟨caddset⟨caddset⟨one, one⟩, one⟩,
    caddset⟨caddset⟨one, one⟩, one⟩⟩ =
    caddset⟨caddset⟨caddset ⟨caddset
    ⟨caddset⟨caddset⟨caddset⟨caddset⟨one, one⟩, one⟩, one⟩, one⟩,
    one⟩, one⟩, one⟩, one⟩
    by (rule MMIsar0.MMI_3t3e9)
  thus 3·3 = 9 by simp
qed
```

107.3 Infimum and supremum in real numbers

Real numbers form a complete ordered field. Here we import a couple of Metamath theorems about supremu and infimum.

If a set S has a smallest element, then the infimum of S belongs to it.

```
lemma (in complex0) lbinfmcl: assumes A1: S ⊆ ℝ and
  A2: ∃x∈S. ∀y∈S. x ≤ y
  shows  Infim(S,ℝ,<) ∈ S
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    S ⊆ real ∧ (∃x∈S. ∀y∈S. ⟨y, x⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real) ⟶
    Sup(S, real,
    converse(lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
    {{complex}} × real)) ∈ S
    by (rule MMIsar0.MMI_lbinfmcl)
  then have
    S ⊆ℝ ∧ ( ∃x∈S. ∀y∈S. x ≤ y) ⟶
    Sup(S,ℝ,converse(<)) ∈ S by simp
  with A1 A2 show thesis using Infim_def by simp
qed
```

Supremum of any subset of reals that is bounded above is real.

```
lemma (in complex0) sup_is_real:
  assumes A ⊆ ℝ  and A ≠ 0 and ∃x∈ℝ. ∀y∈A. y ≤ x
  shows Sup(A,ℝ,<) ∈ ℝ
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real.  ∀y∈A. ⟨x, y⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real) ⟶
    Sup(A, real,
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real) ∈ real
    by (rule MMIsar0.MMI_suprcl)
  with assms show thesis by simp
qed
```

If a real number is smaller that the supremum of A, then we can find an element of A greater than it.

```
lemma (in complex0) suprlub:
  assumes A ⊆ℝ and A ≠ 0 and ∃x∈ℝ. ∀y∈A. y ≤ x
  and B ∈ ℝ  and B < Sup(A,ℝ,<)
  shows ∃z∈A. B < z
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have (A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨x, y⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪
    {{complex}} × real)) ∧ B ∈ real ∧ ⟨B, Sup(A, real,
```

```
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪
    {{complex}} × real)⟩ ∈ lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
    {{complex}} × real ⟶
    (∃z∈A. ⟨B, z⟩ ∈ lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
    {{complex}} × real)
    by (rule MMIsar0.MMI_suprlub)
  with assms show thesis by simp
qed
```

Something a bit more interesting: infimum of a set that is bounded below is real and equal to the minus supremum of the set flipped around zero.

```
lemma (in complex0) infmsup:
  assumes A ⊆ ℝ and A ≠ 0 and ∃x∈ℝ. ∀y∈A. x ≤ y
  shows
  Infim(A,ℝ,<) ∈ ℝ
  Infim(A,ℝ,<) = ( -Sup({z ∈ ℝ. (-z) ∈ A },ℝ,<) )
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have I: MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have
    A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨y, x⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪
    {{complex}} × real) ⟶ Sup(A, real, converse
    (lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪
    {{complex}} × real)) =
    ⋃{x ∈ complex . caddset
    ⟨Sup({z ∈ real . ⋃{x ∈ complex . caddset⟨z, x⟩ = zero} ∈ A}, real,
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real), x⟩ = zero}
    by (rule MMIsar0.MMI_infmsup)
  then have A ⊆ℝ ∧ ¬(A = 0) ∧ ( ∃x∈ℝ. ∀y∈A. x ≤ y) ⟶
    Sup(A,ℝ,converse(<)) = ( -Sup({z ∈ ℝ. (-z) ∈ A },ℝ,<) )
    by simp
  with assms show
    Infim(A,ℝ,<) = ( -Sup({z ∈ ℝ. (-z) ∈ A },ℝ,<) )
```

```
    using Infim_def by simp
  from I have
    A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨y, x⟩ ∉
    lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪
    {{complex}} × real) ⟶ Sup(A, real, converse
    (lessrrel ∩ real × real ∪ {⟨{complex}, complex⟩} ∪
    real × {complex} ∪ {{complex}} × real)) ∈ real
    by (rule MMIsar0.MMI_infmrcl)
  with assms show Infim(A,ℝ,<) ∈ ℝ
    using Infim_def by simp
qed

end
```

References

[1] N. A'Campo. A natural construction for the real numbers. 2003.

[2] R. D. Arthan. The Eudoxus Real Numbers. 2004.

[3] R. Street at al. The Efficient Real Numbers. 2003.

[4] Strecker G.E. Herrlich H. When is ℕ lindelöf? *Comment. Math. Univ. Carolinae*, 1997.

[5] I. L. Reilly and M. K. Vamanamurthy. Some topological anti-properties. *Illinois J. Math.*, 24:382–389, 1980.

[6] D. Zelinski. On Ordered Loops. *Amer. J. Math.*, 70(4):681–697, Oct. 1948.