IsarMathLib

Sławomir Kołodyński, Daniel de la Concepción Sáez

May 26, 2024

Abstract

This is the proof document of the IsarMathLib project version 1.30.0. IsarMathLib is a library of formalized mathematics for Isabelle2024 (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 $\longleftrightarrow$ 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 $\longrightarrow$ 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*⟩

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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)`
  ⟨*proof*⟩

Constructions from the same sets are the same. It is suprising but we do have to use this as a rule in rarte cases.

lemma `same_constr:` **assumes** `x=y` **shows** `P(x) = P(y)`
  ⟨*proof*⟩

Equivalence relations are symmetric.

lemma `equiv_is_sym:` **assumes** `A1: equiv(X,r)` **and** `A2: ⟨x,y⟩ ∈ r`
  shows `⟨y,x⟩ ∈ r`
⟨*proof*⟩

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.

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` ⟨*proof*⟩

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

lemma collection_contain: **assumes** $A \subseteq B$ **shows** $\bigcup A \subseteq \bigcup B$
⟨*proof*⟩

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

lemma ZF1_1_L1: **assumes** $C \neq 0$ **and** $\forall y \in C.\ b(y) = A$
shows $(\bigcup y \in C.\ b(y)) = A$ ⟨*proof*⟩

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

lemma ZF1_1_L2: **assumes** A1: $C \neq 0$ **and** A2: $\forall x \in C.\ b(x) \in A$
and A3: $\forall x\ y.\ x \in C \land y \in C \longrightarrow b(x) = b(y)$
shows $(\bigcup x \in C.\ b(x)) \in A$
⟨*proof*⟩

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: $\forall x \in X. \forall y \in Y.\ a(x,y) = b(x,y)$
shows $\{a(x,y).\ \langle x,y\rangle \in X \times Y\} = \{b(x,y).\ \langle x,y\rangle \in X \times Y\}$
⟨*proof*⟩

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 $p \in X \times Y$ rather than $\langle x,y\rangle \in X \times Y$.

lemma ZF1_1_L4A: **assumes** A1: $\forall x \in X. \forall y \in Y.\ a(\langle x,y\rangle) = b(x,y)$
shows $\{a(p).\ p \in X \times Y\} = \{b(x,y).\ \langle x,y\rangle \in X \times Y\}$
⟨*proof*⟩

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** $U \times V \neq 0$ $U \times V \subseteq X \times Y$ **shows** $U \subseteq X$ **and** $V \subseteq Y$
⟨*proof*⟩

A technical lemma about sections in cartesian products.

lemma section_proj: **assumes** $A \subseteq X \times Y$ **and** $U \times V \subseteq A$ **and** $x \in U$ $y \in V$
shows $U \subseteq \{t \in X.\ \langle t,y\rangle \in A\}$ **and** $V \subseteq \{t \in Y.\ \langle x,t\rangle \in A\}$
⟨*proof*⟩

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

lemma ZF1_1_L4B: **assumes** $\forall x \in X.\ a(x) = b(x)$
shows $\{a(x).\ x \in X\} = \{b(x).\ x \in X\}$
⟨*proof*⟩

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

lemma ZF1_1_L5: **assumes** $X \neq 0$ **and** $\forall x \in X.\ b(x) = c$

shows `{b(x). x∈X} = {c}` ⟨*proof*⟩

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`
  ⟨*proof*⟩

We can choose an element from a nonempty set.

lemma `nonempty_has_element:` **assumes** `X≠0` **shows** `∃x. x∈X`
  ⟨*proof*⟩

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** `⋂0 = 0` ⟨*proof*⟩

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 ≠ 0` **shows** `A≠0`
  ⟨*proof*⟩

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`
  ⟨*proof*⟩

An intersection of subsets is a subset.

lemma `ZF1_1_L7:` **assumes** `A1: I≠0` **and** `A2: ∀i∈I. P(i) ⊆ X`
  shows `( ⋂i∈I. P(i) ) ⊆ X`
⟨*proof*⟩

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` ⟨*proof*⟩

Some properties of singletons.

lemma `ZF1_1_L9:` **assumes** `A1: ∃! x. x∈A ∧ φ(x)`

```
  shows
  ∃a. {x∈A. φ(x)} = {a}
  ⋃ {x∈A. φ(x)} ∈ A
  φ(⋃ {x∈A. φ(x)})
⟨proof⟩
```

A simple version of ZF1_1_L9.

```
corollary singleton_extract: assumes ∃! x. x∈A
  shows (⋃ A) ∈ A
⟨proof⟩
```

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⟩
```

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 ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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}
  ⟨proof⟩
```

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} = 0
```

```
  shows A = 0 ∨ A = {x}
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

A basic property of sets defined by comprehension.

```
lemma comprehension: assumes a ∈ {x∈X. p(x)}
  shows a∈X and p(a) ⟨proof⟩
```

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) ⟨proof⟩
```

The inverse of the `comprehension` lemma.

```
lemma mem_cond_in_set: assumes φ(c) and c∈X
  shows c ∈ {x∈X. φ(x)} ⟨proof⟩
```

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

```
lemma image_rel_mono: assumes r⊆s shows r(A) ⊆ s(A)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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 =` $\bigcup$`{(r{x})×(r{x}). x ∈ domain(r)}`
⟨*proof*⟩

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 ⊆` $\bigcup$`{A{x}×A{x}. x ∈ X}`
⟨*proof*⟩

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 ≠ 0`
  shows `⟨x,y⟩ ∈ (W O (R O W))`
⟨*proof*⟩

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` ⟨*proof*⟩

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

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

The family itself is in its supersets.

lemma `superset_gen`: **assumes** `A⊆X A∈`$\mathcal{A}$ **shows** `A ∈ Supersets(X,`$\mathcal{A}$`)`
  ⟨*proof*⟩

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)`
  ⟨*proof*⟩

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)`
  ⟨*proof*⟩

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)`
  ⟨*proof*⟩

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** $\forall$`t`$\in$`X`. φ`(t)` **and** `x`$\in$`X`
shows φ`(x)` ⟨*proof*⟩

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** $\forall$`x`$\in$`X. p(x) = q(x)`
shows `{p(x). x`$\in$`X} = {q(x). x`$\in$`X}` **and** `{`⟨`x,p(x)`⟩`. x`$\in$`X} = {`⟨`x,q(x)`⟩`. x`$\in$`X}`
⟨*proof*⟩

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`$\neq$`0 X`$\subseteq$`Y` **and** $\forall$`x`$\in$`X. P(x)`
shows `{x`$\in$`Y. P(x)}` $\neq$ `0` ⟨*proof*⟩

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}`$\times$`{y} = {z}`
⟨*proof*⟩

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** $\setminus$ `65`) **where** `A`$\setminus$`B` $\equiv$ `A-B`

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` `(`$\emptyset$`)` **where** $\emptyset$ $\equiv$ `0`

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⟩
```

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⟩
```

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

```
lemma succ_explained: shows succ(n) = n ∪ {n}
  ⟨proof⟩
```

The singleton containing the empty set is a natural number.

```
lemma one_is_nat: shows {0} ∈ nat {0} = succ(0) {0} = 1
⟨proof⟩
```

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 ⟨proof⟩
```

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⟩
```

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*⟩

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`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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`
  ⟨*proof*⟩

For natural numbers membership and inequality are the same and $k \leq n$ is the same as $k \in \text{succ}(n)$. 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)
  ⟨proof⟩
```

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

```
lemma leq_mem_succ: shows k≤n ⟷ k < succ(n) ⟨proof⟩
```

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
  ⟨proof⟩
```

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

```
lemma nat_union_nat: shows nat = ⋃ nat
  ⟨proof⟩
```

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

```
lemma nat_subset_nat: assumes A1: n ∈ nat shows n ⊆ nat
⟨proof⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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))
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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
  ⟨*proof*⟩

Adding and subtracting a natural number cancel each other.

lemma add_subctract: **assumes** m∈nat **shows** (m #+ n) #- n = m
  ⟨*proof*⟩

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) ⟨*proof*⟩

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*⟩

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*⟩

4.2 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*⟩

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*⟩

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)
⟨proof⟩
```

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) ≡ ∃M∈A.∀x∈A. ⟨x,M⟩ ∈ r
```

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

definition

```
HasAminimum(r,A) ≡ ∃m∈A.∀x∈A. ⟨m,x⟩ ∈ r
```

Definition of the maximum of a set.

definition

```
Maximum(r,A) ≡ THE M. M∈A ∧ (∀x∈A. ⟨x,M⟩ ∈ r)
```

Definition of a minimum of a set.

definition

```
Minimum(r,A) ≡ 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) ≡ 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) ≡ 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) ≡ HasAmaximum(r,⋂a∈A. r-{a})
```

Infimum is defined analogously.

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

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 ⟨proof⟩
```

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 ⟨proof⟩
```

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 ⟨proof⟩
```

A total relation is reflexive.

```
lemma total_is_refl:
  assumes r {is total on} X
  shows refl(X,r) ⟨proof⟩
```

A linear order is partial order.

```
lemma Order_ZF_1_L2: assumes IsLinOrder(X,r)
  shows IsPartOrder(X,r)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

We can restrict a partial order relation to the domain.

```
lemma part_ord_restr: assumes IsPartOrder(X,r)
  shows IsPartOrder(X,r ∩ X×X)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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}
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

`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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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 ⟨proof⟩
```

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 ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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)`
  ⟨*proof*⟩

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*⟩

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 ($\leq$ 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩

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 maximuma 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`
  ⟨*proof*⟩

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

lemma `set_min_not_empty:` **assumes** `HasAminimum(r,A)` **shows** `A≠0`
  ⟨*proof*⟩

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*⟩

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

lemma `set_inf_not_empty:` **assumes** `HasAnInfimum(r,A)` **shows** `A≠0`
⟨*proof*⟩

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*⟩

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⟩

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⟩

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⟩

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⟩

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⟩

Set that has a maximum is bounded above.

```
lemma Order_ZF_4_L7:
  assumes HasAmaximum(r,A)
  shows IsBoundedAbove(A,r)
```
  ⟨proof⟩

Set that has a minimum is bounded below.

```
lemma Order_ZF_4_L8A:
  assumes HasAminimum(r,A)
  shows IsBoundedBelow(A,r)
```
  ⟨proof⟩

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})
```
  ⟨proof⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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})`
  ⟨*proof*⟩

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})`
  ⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩

end
```

8 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.

8.1 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`
⟨*proof*⟩

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)`
⟨*proof*⟩

A lemma explains what `fstdom` is for.

lemma `fstdomdef`: **assumes** `A1: f: X×Y → Z` **and** `A2: Y≠∅`
shows `fstdom(f) = X`

⟨*proof*⟩

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*⟩

A variant of `func1_1_L1A`.

lemma `func1_1_L1B`: **assumes** `A1: f:X→Y` **and** `A2: Y⊆Z`
shows `f:X→Z`
⟨*proof*⟩

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*⟩

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

lemma `vimage_converse`: **shows** `r-(A) = converse(r)(A)`
⟨*proof*⟩

The image is the inverse image of converse.

lemma `image_converse`: **shows** `converse(r)-(A) = r(A)`
⟨*proof*⟩

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

lemma `vimage_comp`: **shows** `(r O s)-(A) = s-(r-(A))`
⟨*proof*⟩

A version of `vimage_comp` for three functions.

lemma `vimage_comp3`: **shows** `(r O s O t)-(A) = t-(s-(r-(A)))`
⟨*proof*⟩

Inverse image of any set is contained in the domain.

lemma `func1_1_L3`: **assumes** `A1: f:X→Y` **shows** `f-(D) ⊆ X`
⟨*proof*⟩

The inverse image of the range is the domain.

lemma `func1_1_L4`: **assumes** `f:X→Y` **shows** `f-(Y) = X`
⟨*proof*⟩

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*⟩

Function is a subset of cartesian product.

lemma `fun_subset_prod`: **assumes** `A1: f:X→Y` **shows** `f ⊆ X×Y`
⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

Image of a greater set is greater.

lemma `func1_1_L8`: **assumes** `A1: A⊆B` **shows** `f(A)⊆ f(B)`
⟨*proof*⟩

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⟩
```

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*⟩

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⟩
```

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 ⟨proof⟩
```

A set defined by a lambda-type expression is a fuction. 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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*⟩

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*⟩

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)`
  ⟨*proof*⟩

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}`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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≠∅
  ⟨proof⟩
```

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≠∅
  ⟨proof⟩
```

What is the inverse image of a singleton?

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

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 ⟨proof⟩
```

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⟩
```

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⟩
```

Function defined on a singleton is a single pair.

```
lemma func_singleton_pair: assumes A1: f : {a}→X
  shows f = {⟨a, f(a)⟩}
⟨proof⟩
```

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⟩
```

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

```
lemma pair_val: shows {⟨x,y⟩}(x) = y
  ⟨proof⟩
```

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⟩
```

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⟩
```

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}`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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`
  ⟨*proof*⟩

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`
  ⟨*proof*⟩

What is the image of a singleton?

lemma singleton_image:
  assumes $f \in X \to Y$ **and** $x \in X$
  shows f{x} = {f(x)}
  ⟨*proof*⟩

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** $f:X \to Y$ $x \in A$ $A \subseteq X$
  shows $f(x) \in f(A)$
  ⟨*proof*⟩

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: $f:X \to Y$ **shows** f(X) = range(f)
⟨*proof*⟩

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

lemma diff_image_diff: **assumes** A1: $f: X \to Y$ **and** A2: $A \subseteq X$
  shows $f(X) \setminus f(A) \subseteq f(X \setminus A)$
⟨*proof*⟩

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

lemma image_of_Inter: **assumes** A1: $f:X \to Y$ **and**
  A2: $I \neq \emptyset$ **and** A3: $\forall i \in I.\ P(i) \subseteq X$
  shows $f(\bigcap i \in I.\ P(i)) \subseteq (\bigcap i \in I.\ f(P(i)))$
⟨*proof*⟩

The image of union is the union of images.

lemma image_of_Union: **assumes** A1: $f:X \to Y$ **and** A2: $\forall A \in M.\ A \subseteq X$
  shows $f(\bigcup M) = \bigcup \{f(A).\ A \in M\}$
⟨*proof*⟩

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

lemma codomain_nonempty: **assumes** $f:X \to Y$ $X \neq \emptyset$ **shows** $Y \neq \emptyset$
  ⟨*proof*⟩

The image of a nonempty subset of domain is nonempty.

lemma func1_1_L15A:
  assumes A1: $f: X \to Y$ **and** A2: $A \subseteq X$ **and** A3: $A \neq \emptyset$
  shows $f(A) \neq \emptyset$
⟨*proof*⟩

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 $f:X \to Y$ **and** $A \subseteq X$ **and** $\forall y \in f(A).\ P(y)$

```
  shows ∀x∈A. P(f(x))
  ⟨proof⟩
```

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⟩
```

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

```
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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

8.2 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

A function restricted to its domain is itself.

```
lemma restrict_domain: assumes f:X→Y
  shows restrict(f,X) = f
⟨proof⟩
```

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⟩
```

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⟩
```

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

```
lemma right_comp_id_any: shows r O id(C) = restrict(r,C)
  ⟨proof⟩
```

8.3 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⟩
```

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⟩
```

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}
  ⟨proof⟩
```

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⟩
```

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⟩
```

8.4 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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 $f : \{0,1\} \rightarrow \mathbb{N}, f(n) = n+1$ and $y = 3$. Then $f^{-1} : \{1,2\} \rightarrow \{0,1\}$ 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*⟩

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*⟩

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*⟩

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))`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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})`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

8.5 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 ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

end

9 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.

9.1 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

Join is associative.

```
lemma (in join_semilatt) join_assoc: assumes x∈L y∈L z∈L
  shows x⊔(y⊔z) = x⊔y⊔z
⟨proof⟩
```

Join is idempotent.

```
lemma (in join_semilatt) join_idempotent: assumes x∈L shows x⊔x = x

  ⟨proof⟩
```

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*⟩

Meet is associative.

lemma (**in** meet_semilatt) meet_assoc: **assumes** x∈L y∈L z∈L
  shows x⊓(y⊓z) = x⊓y⊓z
⟨*proof*⟩

Meet is idempotent.

lemma (**in** meet_semilatt) meet_idempotent: **assumes** x∈L **shows** x⊓x = x

⟨*proof*⟩

end

10 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.

10.1 Order on natural numbers

This is the only section in this theory.

To prove that $\leq$ is a total order, we use a result on ordinals.

lemma NatOrder_ZF_1_L1:
  assumes a∈nat **and** b∈nat
  shows a ≤ b ∨ b ≤ a
⟨*proof*⟩

$\leq$ 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*⟩

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*⟩

end

11 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.

11.1 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*⟩

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*⟩

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))
```
⟨*proof*⟩

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*⟩

11.2 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 ”·” 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⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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⟩
```

11.3 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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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 ⟨proof⟩
```

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
  ⟨proof⟩
```

11.4 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)
  ⟨proof⟩
```

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⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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⟩
```

11.5 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⟩
```

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⟩
```

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⟩
```

Another trivial fact: identity is the only bijection of a singleton with itself.

```
lemma single_bij_id: shows bij({x},{x}) = {id({x})}
⟨proof⟩
```

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⟩
```

11.6 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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

11.7 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: A_r = restrict(A,Y×Y) M_r = 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⟩
```

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⟩
```

end

12 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

12.1 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⟩
```

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⟩
```

Identity is an order isomorphism.

```
lemma id_ord_iso: shows id(X) ∈ ord_iso(X,r,X,r)
```

⟨*proof*⟩

Identity is the only order automorphism of a singleton.

```
lemma id_ord_auto_singleton:
  shows ord_iso({x},r,{x},r) = {id({x})}
  ⟨proof⟩
```

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⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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*⟩

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*⟩

12.2 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*⟩

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*⟩

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⟩
```

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⟩
```

12.3 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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

end

13 Finite sets - introduction

theory `Finite_ZF` **imports** `ZF1 Nat_ZF_IML ZF.Cardinal`

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|`
  ⟨*proof*⟩

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*⟩

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`
⟨*proof*⟩

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*⟩

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})`
⟨*proof*⟩

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*⟩

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)`
⟨*proof*⟩

Natural numbers are finite subsets of the set of natural numbers.

lemma `nat_finpow_nat`: **assumes** `n ∈ nat` **shows** `n ∈ FinPow(nat)`
⟨*proof*⟩

A finite subset is a finite subset of itself.

lemma `fin_finpow_self`: **assumes** `A ∈ FinPow(X)` **shows** `A ∈ FinPow(A)`
⟨*proof*⟩

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`
⟨*proof*⟩

Induction for finite powerset. This is smilar to the standard Isabelle's `Fin_induct`.

theorem `FinPow_induct`: **assumes** `A1: P(0)` **and**

A2: $\forall$A $\in$ FinPow(X). P(A) $\longrightarrow$ ($\forall$a$\in$X. P(A $\cup$ {a})) **and**
A3: B $\in$ FinPow(X)
shows P(B)
⟨*proof*⟩

A subset of a finite subset is a finite subset.

lemma subset_finpow: **assumes** A $\in$ FinPow(X) **and** B $\subseteq$ A
shows B $\in$ FinPow(X)
⟨*proof*⟩

If we subtract anything from a finite set, the resulting set is finite.

lemma diff_finpow:
assumes A $\in$ FinPow(X) **shows** A-B $\in$ FinPow(X)
⟨*proof*⟩

If we remove a point from a finites subset, we get a finite subset.

corollary fin_rem_point_fin: **assumes** A $\in$ FinPow(X)
shows A - {a} $\in$ FinPow(X)
⟨*proof*⟩

Cardinality of a nonempty finite set is a successsor of some natural number.

lemma card_non_empty_succ:
assumes A1: A $\in$ FinPow(X) **and** A2: A $\neq$ 0
shows $\exists$n $\in$ nat. |A| = succ(n)
⟨*proof*⟩

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 $\in$ FinPow(X) **and** A $\neq$ 0
shows |A| $\neq$ 0
⟨*proof*⟩

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: $\forall$k$\in$nat.
($\forall$A $\in$ FinPow(X). A $\lesssim$ k $\longrightarrow$ P(A)) $\longrightarrow$
($\forall$A $\in$ FinPow(X). A $\lesssim$ succ(k) $\longrightarrow$ P(A))
and A3: A $\in$ FinPow(X) **shows** P(A)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

Empty set is in finite power set.

lemma empty_in_finpow: **shows** 0 ∈ FinPow(X)
  ⟨*proof*⟩

Singleton is in the finite powerset.

lemma singleton_in_finpow: **assumes** x ∈ X
  shows {x} ∈ FinPow(X) ⟨*proof*⟩

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)
  ⟨*proof*⟩

Union of finite number of finite sets is finite.

```
lemma fin_union_finpow: assumes M ∈ FinPow(FinPow(X))
  shows ⋃M ∈ FinPow(X)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

Singletons are in the finite powerset.

```
lemma Finite1_L16: assumes x∈X shows {x} ∈ Fin(X)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

Finite range functions have finite range.

```
lemma Finite1_L18: assumes f ∈ FinRangeFunctions(X,Y)
  shows {f(x). x∈X} ∈ Fin(Y)
  ⟨proof⟩
```

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)
```

⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

The empty set is the only order automorphism of itself.

lemma `empty_ord_iso_empty`: **shows** `ord_iso(0,r,0,R) = {0}`
⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

end

17 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.

17.1 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*⟩

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`
  ⟨*proof*⟩

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*⟩

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`
  ⟨*proof*⟩

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` ⟨*proof*⟩

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*⟩

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⟩
```

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⟩
```

Equivalence classes are not empty.

```
lemma EquivClass_1_L5:
  assumes A1: refl(A,r) and A2: C ∈ A//r
  shows C≠0
⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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:` C_1 `∈ A//r` C_2 `∈ A//r`
  and `A4:` z_1 `∈` C_1`×`C_2 z_2 `∈` C_1`×`C_2
  shows `r{f(`z_1`)} = r{f(`z_2`)}`
⟨*proof*⟩

The values of congruent functions of two variables are in the space.

lemma `EquivClass_1_L8:`
  assumes `A1: equiv(A,r)` **and** `A2:` C_1 `∈ A//r` **and** `A3:` C_2 `∈ A//r`
  and `A4: z ∈` C_1`×`C_2 **and** `A5: Congruent2(r,f)`
  shows `f(z) ∈ A`
⟨*proof*⟩

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*⟩

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*⟩

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`
  ⟨*proof*⟩

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*⟩

17.2 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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩⟩ ∧
```

```
  Mp⟨ Ap⟨b,c⟩,a ⟩ = Ap⟨ Mp⟨b,a⟩, Mp⟨c,a⟩⟩
```
⟨*proof*⟩

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*⟩

17.3 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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

end

18 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.

18.1 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)` is the previous natural number and denotes the difference between natural numbers n and k.

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}`
  ⟨*proof*⟩

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*⟩

Codomain of a nonempty list is nonempty.

lemma `nelist_vals_nonempty`: **assumes** `a:succ(n)→Y`
  shows `Y≠0` ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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)`
  ⟨*proof*⟩

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*⟩

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⟩
```

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)
  ⟨proof⟩
```

`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⟩
```

`NELists` are non-empty lists

```
lemma non_zero_List_func_is_NEList:
  shows NELists(X) = {a∈Lists(X). a≠0}
⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

A reformulation of definition of `Init`.

```
lemma init_def: assumes n ∈ nat and a:succ(n)→X
  shows Init(a) = restrict(a,n)
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩})
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

18.2 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 $(n+1) \to X$ of lists of length $n+1$ of elements of X and the cartesian product $(n \to X) \times X$.

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

end

19 Formal languages

theory `Finite_State_Machines_ZF` **imports** `FiniteSeq_ZF Finite1 ZF.CardinalArith`

begin

19.1 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*⟩

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`
  ⟨*proof*⟩

19.2 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,`s_0`,t,F){is an DFSA for alphabet}Σ ≡ Finite(S) ∧` s_0
`∈ 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,s_0,t,F){is an DFSA for alphabet}Σ ⟹
{reduce D-relation}(S,s_0,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,s_0,t){in alphabet}Σ), we add the extra condition that ⟨⟨∅, s_0⟩, ∅, s_0⟩ is also a valid transition.

definition
DFSASatisfy (_ <-D '(_,_,_,_')\{in alphabet\}_) **where**
Finite(Σ) ⟹ (S,s_0,t,F){is an DFSA for alphabet}Σ ⟹ i∈Lists(Σ) ⟹

i <-D (S,s_0,t,F){in alphabet}Σ ≡ (∃q∈F. ⟨⟨i,s_0⟩,⟨0,q⟩⟩∈ ({reduce D-relation}(S,s_0,t){in alphabet}Σ)^*) ∨ (i = 0 ∧ s_0∈F)

We define a locale for better notation

locale DetFinStateAuto =
fixes S **and** s_0 **and** t **and** F **and** Σ
assumes finite_alphabet: Finite(Σ)

assumes DFSA: (S,s_0,t,F){is an DFSA for alphabet}Σ

We abbreviate the reduce relation to a single symbol within this locale.

abbreviation (**in** DetFinStateAuto) r_D **where**
r_D ≡ {reduce D-relation}(S,s_0,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,s_0,t,F){in alphabet}Σ

Destruction lemma about deterministic finite state automata.

lemma (**in** DetFinStateAuto) DFSA_dest:
shows s_0∈S F⊆S t:S×Σ → S Finite(S) ⟨*proof*⟩

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,s_0,t,F){in alphabet}Σ} {is a language with alphabet}Σ

⟨*proof*⟩

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(rD) function(rD) ⟨proof⟩
```

The relation, that is actually a function has the following domain and range:

```
lemma (in DetFinStateAuto) reduce_function:
shows rD:NELists(Σ)×S→Lists(Σ)×S
⟨proof⟩
```

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(rD) ⊆ Lists(Σ)×S NELists(Σ)×S ⊆ field(rD)
⟨proof⟩
```

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⟩⟩∈rD^*
  shows restrict(w,domain(u)) = u
⟨proof⟩
```

```
lemma (in DetFinStateAuto) relation_deteministic:
  assumes ⟨⟨w,s⟩,⟨u,v⟩⟩∈rD^* ⟨⟨w,s⟩,⟨u,m⟩⟩∈rD^*
  shows v=m
⟨proof⟩
```

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,s0⟩,⟨0,q⟩⟩ ∈ rD^*
⟨proof⟩
```

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` ⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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}
  ⟨proof⟩

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
  ⟨proof⟩
```

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 ⟨proof⟩
```

19.3 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⟩
```

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⟩
```

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⟩
```

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*⟩

19.4 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
⟹ 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}

19.5 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 $\equiv$ {⟨⟨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 $\equiv$ 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 $\neq$ 0} Σ
  ⟨*proof*⟩

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*⟩

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*⟩

Since both are relations, they are equal

corollary (**in** NonDetFinStateAuto) relation_NFSA_to_DFSA:
  shows r_N = rPow ⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

end

20 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.

20.1 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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

20.2 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*⟩

20.3 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*⟩

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*⟩

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*⟩

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*⟩

20.4 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 $f : X \times n \rightarrow X \times n$, $f\langle x,k\rangle = \langle f_k(x), k+1\rangle$ 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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

20.5 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*⟩

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*⟩

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)`
  ⟨*proof*⟩

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))`
  ⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

end

21 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.

21.1 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)
```

21.2 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)
  ⟨proof⟩
```

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*⟩

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*⟩

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*⟩

What is the enumeration of the empty set?

```
lemma (in enums) empty_enum: shows σ(0) = 0
```
⟨*proof*⟩

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*⟩

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*⟩

end

22 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.

22.1 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

Another way of formulating information contained in `fold_append` is to start with a longer sequence $a : n+1 \rightarrow X$ 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⟩
```

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⟩
```

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⟩

end
```

23 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.

23.1 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
```
⟨*proof*⟩

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)` ⟨*proof*⟩

Bisection of X is a pair of subsets of X.

lemma `bisec_is_pair:` **assumes** `Q ∈ Bisections(X)`
shows `Q = ⟨fst(Q), snd(Q)⟩`
⟨*proof*⟩

The set of bisections of the empty set is empty.

lemma `bisec_empty:` **shows** `Bisections(0) = 0`
⟨*proof*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

23.2 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⟩
```

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⟩

end
```

24 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.

24.1 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 inverse 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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

end

25 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.

25.1 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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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*⟩

end

26 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].

26.1 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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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*⟩

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*⟩

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
⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

The definition of the nonnegative set in the notation used in the `loop1` locale:

lemma (in loop1) nonneg_definition:
shows x ∈ L^+ ⟷ **0** ≤ x ⟨*proof*⟩

The nonnegative set is contained in the loop.

lemma (in loop1) nonneg_subset: **shows** L^+ ⊆ L
⟨*proof*⟩

The positive set is contained in the loop.

lemma (in loop1) positive_subset: **shows** L_+ ⊆ L
⟨*proof*⟩

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**)

⟨*proof*⟩

Another form of the definition of the positive set in the notation used in the `loop1` locale:

lemma (**in** `loop1`) `posset_definition1`:
shows $x \in L_+ \longleftrightarrow \mathbf{0}<x$
⟨*proof*⟩

The order in an ordered loop is antisymmeric.

lemma (**in** `loop1`) `loop_ord_antisym`: **assumes** $x\leq y$ **and** $y\leq x$
shows $x=y$
⟨*proof*⟩

The loop order is transitive.

lemma (**in** `loop1`) `loop_ord_trans`: **assumes** $x\leq y$ **and** $y\leq z$ **shows** $x\leq z$
⟨*proof*⟩

The loop order is reflexive.

lemma (**in** `loop1`) `loop_ord_refl`: **assumes** $x\in L$ **shows** $x\leq x$
⟨*proof*⟩

A form of mixed transitivity for the strict order:

lemma (**in** `loop1`) `loop_strict_ord_trans`: **assumes** $x\leq y$ **and** $y<z$
shows $x<z$
⟨*proof*⟩

Another form of mixed transitivity for the strict order:

lemma (**in** `loop1`) `loop_strict_ord_trans1`: **assumes** $x<y$ **and** $y\leq z$
shows $x<z$
⟨*proof*⟩

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*⟩

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** $x\leq y$
shows $\mathbf{0} \leq -x+y$ $(-x+y) \in L^+$ $\mathbf{0} \leq y-x$ $(y-x) \in L^+$
⟨*proof*⟩

We can move an element to the other side of a strict inequality.

lemma (**in** `loop1`) `ls_other_side`: **assumes** $x<y$
shows $\mathbf{0} < -x+y$ $(-x+y) \in L_+$ $\mathbf{0} < y-x$ $(y-x) \in L_+$
⟨*proof*⟩

We can add sides of inequalities.

lemma (**in** `loop1`) `add_ineq`: **assumes** `x≤y z≤t`
  shows `x+z ≤ y+t`
⟨*proof*⟩

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*⟩

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*⟩

Subtracting a positive element decreases the value.

lemma (**in** `loop1`) `subtract_pos`: **assumes** `x∈L` **0**`<y`
  shows `x-y < x` **and** `(-y+x) < x`
⟨*proof*⟩

end

27 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.

27.1 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)`
⟨*proof*⟩

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`
⟨*proof*⟩

Semigroup operation is closed.

lemma (**in** `semigr0) semigr_closed:`

```
  assumes a∈G  b∈G shows a·b ∈ G
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

What is the product of one element list?

```
lemma (in semigr0) prod_of_1elem: assumes A1: a: 1 → G
  shows (∏ a) = a(0)
⟨proof⟩
```

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⟩
```

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⟩
```

$a \cdot b \cdot (c \cdot d) = a \cdot (b \cdot c) \cdot d$ 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*⟩

27.2 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 ∏(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)
  ⟨*proof*⟩

Definition of product over a set expressed in notation of the semigr0 locale.

lemma (**in** semigr1) setproddef:
  shows ∏(Λ,a) = ∏ (a O σ(Λ))
  ⟨*proof*⟩

A composition of enumeration of a nonempty finite subset of 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*⟩

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
  ⟨*proof*⟩

What is product over a singleton?

lemma (**in** semigr1) gen_prod_singleton:
  assumes A1: x ∈ X
  shows ∏({x},a) = a(x)
⟨*proof*⟩

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*⟩

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⟩
```

27.3 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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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))
  ⟨proof⟩

end
```

28 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.

28.1 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)})
  ⟨proof⟩
```

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⟩
```

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⟩

end
```

29 Monoids

```
theory Monoid_ZF imports func_ZF Loop_ZF Semigroup_ZF

begin
```

This theory provides basic facts about monoids.

29.1 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)`
  ⟨*proof*⟩

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`
  ⟨*proof*⟩

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*⟩

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*⟩

The monoid carrier is not empty.

```
lemma (in monoid0) group0_1_L3A: shows G≠0
⟨proof⟩
```

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
  ⟨proof⟩
```

The range of the monoid operation is the whole monoid carrier.

```
lemma (in monoid0) group0_1_L3B: shows range(f) = G
⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

The monoid operation is associative.

```
lemma (in monoid0) sum_associative:
  assumes a∈G b∈G c∈G
  shows (a⊕b)⊕c = a⊕(b⊕c)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

end

30 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 $n \cdot x, n \in \mathbb{N}$).

30.1 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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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)
  ⟨*proof*⟩

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*⟩

30.2 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.

The zero's multiple of a monoid element is its neutral element.

lemma (**in** monoid1) nat_mult_zero: **shows** 0·x = **0** ⟨*proof*⟩

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 ⟨*proof*⟩

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*⟩

One element of a monoid is that element.

lemma (**in** monoid1) nat_mult_one: **assumes** x∈G **shows** 1·x = x
⟨*proof*⟩

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*⟩

end

31 Groups - introduction

theory Group_ZF **imports** Monoid_ZF

begin

This theory file covers basics of group theory.

31.1 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)}
```

We will use the miltiplicative notation for groups. The neutral element is denoted 1.

```
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)
```

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)
  ⟨proof⟩
```

The theorems proven in the `monoid` context are valid in the `group0` context.

```
sublocale group0 < monoid: monoid0 G P groper
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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))
```

⟨*proof*⟩

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 ⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

The inverse is in the group.

lemma (**in** group0) inverse_in_group: **assumes** A1: x∈G **shows** x^{-1}∈G
⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

The inverse of the neutral element is the neutral element.

```
lemma (in group0) group_inv_of_one: shows 1⁻¹ = 1
  ⟨proof⟩
```

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⟩
```

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 ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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*⟩

What is the inverse of a product?

lemma (in group0) group_inv_of_two:
  assumes A1: a∈G **and** A2: b∈G
  shows $b^{-1}\cdot a^{-1} = (a\cdot b)^{-1}$
⟨*proof*⟩

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
  $(a\cdot b\cdot c)^{-1} = c^{-1}\cdot(a\cdot b)^{-1}$
  $(a\cdot b\cdot c)^{-1} = c^{-1}\cdot(b^{-1}\cdot a^{-1})$
  $(a\cdot b\cdot c)^{-1} = c^{-1}\cdot b^{-1}\cdot a^{-1}$
⟨*proof*⟩

The inverse of the inverse is the element.

lemma (in group0) group_inv_of_inv:
  assumes a∈G **shows** $a = (a^{-1})^{-1}$
  ⟨*proof*⟩

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*⟩

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*⟩

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
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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⟩
```

The group inverse is onto.

```
lemma (in group0) group_inv_surj: shows GroupInv(G,P)(G) = G
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

31.2 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)
⟨*proof*⟩

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*⟩

A subgroup is contained in the group.

lemma **(in** group0) group0_3_L2:
  assumes A1: IsAsubgroup(H,P)
  shows H ⊆ G
⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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}
  ⟨*proof*⟩

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*⟩

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*⟩

The singleton with the neutral element is a subgroup.

corollary (**in** group0) unit_singl_subgr:

```
  shows IsAsubgroup({1},P)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

31.3 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*⟩

A group is a quasigroup and a loop.

lemma (in group0) group_is_loop: **shows** IsAquasigroup(G,P) **and** IsAloop(G,P)
⟨*proof*⟩

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*⟩

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*⟩

end

32 Groups 1

theory Group_ZF_1 **imports** Group_ZF

begin

In this theory we consider right and left translations and odd functions.

32.1 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⟩
```

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
  ⟨proof⟩
```

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
  T_g(T_h(a)) = g·h·a
  T_g(T_h(a)) = LeftTranslation(G,P,g·h)(a)
⟨proof⟩
```

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: T_g = RightTranslation(G,P,g)  T_h = RightTranslation(G,P,h)
  shows
 T_g(T_h(a)) = a·h·g
  T_g(T_h(a)) = RightTranslation(G,P,h·g)(a)
⟨proof⟩
```

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⟩
```

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: T_g = LeftTranslation(G,P,g) T_h = LeftTranslation(G,P,h)
shows $T_g(T_h(A))$ = LeftTranslation(G,P,g·h)(A)
⟨*proof*⟩

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: T_g = RightTranslation(G,P,g) T_h = RightTranslation(G,P,h)
  shows $T_g(T_h(A))$ = {a·h·g. a∈A}
⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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*⟩

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*⟩

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

32.2 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 )
  ⟨proof⟩
```

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⟩
```

32.3 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}
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

end

33 Groups - and 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.

33.1 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⟩
```

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*⟩

end

34 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.

34.1 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 prove them by hand.

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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
  $a\cdot b^{-1}\cdot c^{-1}\cdot d^{-1} = a\cdot d^{-1}\cdot b^{-1}\cdot c^{-1}$
  $a\cdot b\cdot c\cdot d^{-1} = a\cdot d^{-1}\cdot b\cdot c$
  $a\cdot b\cdot c^{-1}\cdot d^{-1} = a\cdot c^{-1}\cdot (b\cdot d^{-1})$
⟨*proof*⟩

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
  $a\cdot b\cdot c\cdot d^{-1} = a\cdot d^{-1}\cdot b\cdot c$
  $a\cdot d\cdot (b\cdot d\cdot (c\cdot d))^{-1} = a\cdot (b\cdot c)^{-1}\cdot d^{-1}$
  $a\cdot (b\cdot c)\cdot d = a\cdot b\cdot d\cdot c$
⟨*proof*⟩

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
  $a\cdot (b\cdot c)^{-1} = (a\cdot d^{-1}\cdot c^{-1})\cdot (d\cdot b^{-1})$
  $a\cdot b\cdot (c\cdot d) = c\cdot a\cdot (b\cdot d)$
  $a\cdot b\cdot (c\cdot d) = a\cdot c\cdot (b\cdot d)$
  $a\cdot (b\cdot c^{-1})\cdot d = a\cdot b\cdot d\cdot c^{-1}$
  $(a\cdot b)\cdot (c\cdot d)^{-1}\cdot (b\cdot d^{-1})^{-1} = a\cdot c^{-1}$
⟨*proof*⟩

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
  $a\cdot b^{-1}\cdot (c\cdot d^{-1}) = a\cdot c\cdot (b^{-1}\cdot d^{-1})$
  $a\cdot b^{-1}\cdot (c\cdot d^{-1}) = a\cdot c\cdot b^{-1}\cdot d^{-1}$
⟨*proof*⟩

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⟩
```

end

35 Groups 2

```
theory Group_ZF_2 imports AbelianGroup_ZF func_ZF EquivClass1
```

begin

This theory continues Group_ZF.thy and considers lifting the group structure to function spaces and projecting the group structure to quotient spaces, in particular the quotient qroup.

35.1 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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*⟩

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

35.2 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

35.3 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
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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})
  ⟨proof⟩
```

35.4 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⟩
```

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⟩
```

end

36 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.

36.1 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*⟩

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*⟩

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)
  ⟨*proof*⟩

The set of finite range functions is s 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*⟩

36.2 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))$^{-1}$
⟨*proof*⟩

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*⟩

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 $(\text{HomDiff}(G, f))$. 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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
    ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

36.3 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*⟩

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*⟩

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))
```
⟨*proof*⟩

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))
```
⟨*proof*⟩

The ”almost equal” relation is symmetric.

```
lemma (in group1) Group_ZF_3_3_L3A: assumes A1: s≅r
  shows r≅s
```
⟨*proof*⟩

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

36.4 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⟩)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

36.5 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⟩

end
```

37 Direct product

```
theory DirectProduct_ZF imports func_ZF

begin
```

This theory considers the direct product of binary operations. Contributed by Seo Sanghyeon.

37.1 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⟩
```

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)⟩⟩
  ⟨proof⟩
```

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
  ⟨proof⟩
```

37.2 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⟩
```

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⟩
```

end

38 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.

38.1 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. $\texttt{G}^+$ denotes the set of nonnegative elements (that satisfy $1 \leq a$) and $\texttt{G}_+$ 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)`
⟨*proof*⟩

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`
⟨*proof*⟩

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
  ⟨*proof*⟩

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**)
  ⟨*proof*⟩

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**
⟨*proof*⟩

The group order is reflexive.

lemma (**in** group3) OrderedGroup_ZF_1_L3: **assumes** g∈G
  shows g≤g
  ⟨*proof*⟩

1 is nonnegative.

lemma (**in** group3) OrderedGroup_ZF_1_L3A: **shows** **1**∈G$^+$
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

It is good to have transitivity handy.

lemma (**in** group3) Group_order_transitive:
  assumes A1: a≤b b≤c **shows** a≤c
⟨*proof*⟩

The order in an ordered group is antisymmetric.

lemma (**in** group3) group_order_antisym:
  assumes A1: a≤b b≤a **shows** a=b
⟨*proof*⟩

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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*⟩

The nonnegative set is contained in the group.

lemma (in group3) OrderedGroup_ZF_1_L4E: **shows** $G^+ \subseteq G$
⟨*proof*⟩

The positive set is contained in the nonnegative set, hence in the group.

lemma (in group3) pos_set_in_gr: **shows** $G_+ \subseteq G^+$ **and** $G_+ \subseteq G$
⟨*proof*⟩

Taking the inverse on both sides reverses the inequality.

lemma (in group3) OrderedGroup_ZF_1_L5:
assumes A1: $a \leq b$ **shows** $b^{-1} \leq a^{-1}$
⟨*proof*⟩

If an element is smaller that the unit, then its inverse is greater.

lemma (in group3) OrderedGroup_ZF_1_L5A:
assumes A1: $a \leq \mathbf{1}$ **shows** $\mathbf{1} \leq a^{-1}$
⟨*proof*⟩

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: $a \in G$ **and** A2: $\mathbf{1} \leq a^{-1}$
shows $a \leq \mathbf{1}$
⟨*proof*⟩

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: $\mathbf{1} \leq a$ **shows** $a^{-1} \leq \mathbf{1}$ **and** $\neg(a \leq \mathbf{1} \wedge a \neq \mathbf{1})$
⟨*proof*⟩

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: $\mathbf{1} \leq a$ $\mathbf{1} \leq b$
shows $a^{-1} \leq b$
⟨*proof*⟩

38.2 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: $b \in G$ **and** A2: $a \leq b^{-1}$
shows $b \leq a^{-1}$
⟨*proof*⟩

We can cancel the same element on both sides of an inequality.

lemma (in group3) OrderedGroup_ZF_1_L5AE:
assumes A1: $a \in G$ $b \in G$ $c \in G$ **and** A2: $a \cdot b \leq a \cdot c$
shows $b \leq c$
⟨*proof*⟩

We can cancel the same element on both sides of an inequality, right side.

lemma (in group3) ineq_cancel_right:
assumes $a \in G$ $b \in G$ $c \in G$ **and** $a \cdot b \leq c \cdot b$
shows $a \leq c$
⟨*proof*⟩

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: $a \in G$ $b \in G$ $c \in G$ **and** A2: $a \cdot b^{-1} \leq a \cdot c^{-1}$
shows $c \leq b$
⟨*proof*⟩

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: $a \in G$ **and** A2: $a^{-1} \leq b$
shows $b^{-1} \leq a$
⟨*proof*⟩

We can multiply the sides of two inequalities.

lemma (in group3) OrderedGroup_ZF_1_L5B:
assumes A1: $a \leq b$ **and** A2: $c \leq d$
shows $a \cdot c \leq b \cdot d$
⟨*proof*⟩

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: $c \in G$ **and** A2: $a \leq b \cdot c$ **and** A3: $b \leq b_1$
shows $a \leq b_1 \cdot c$
⟨*proof*⟩

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: $b \in G$ **and** A2: $a \leq b \cdot c$ **and** A3: $c \leq b_1$

shows $a \leq b \cdot b_1$
⟨*proof*⟩

We can replace factors on one side of an inequality with greater ones.

lemma (in group3) OrderedGroup_ZF_1_L5E:
assumes A1: $a \leq b \cdot c$ **and** A2: $b \leq b_1$ $c \leq c_1$
shows $a \leq b_1 \cdot c_1$
⟨*proof*⟩

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: $\mathbf{1} \leq a$ **and** A2: $b \in G$
shows $b \leq a \cdot b$ $b \leq b \cdot a$
⟨*proof*⟩

We can multiply the right hand side of an inequality by a nonnegative element.

lemma (in group3) OrderedGroup_ZF_1_L5G: **assumes** A1: $a \leq b$
and A2: $\mathbf{1} \leq c$ **shows** $a \leq b \cdot c$ $a \leq c \cdot b$
⟨*proof*⟩

We can put two elements on the other side of inequality, changing their sign.

lemma (in group3) OrderedGroup_ZF_1_L5H:
assumes A1: $a \in G$ $b \in G$ **and** A2: $a \cdot b^{-1} \leq c$
shows
$a \leq c \cdot b$
$c^{-1} \cdot a \leq b$
⟨*proof*⟩

We can multiply the sides of one inequality by inverse of another.

lemma (in group3) OrderedGroup_ZF_1_L5I:
assumes $a \leq b$ **and** $c \leq d$
shows $a \cdot d^{-1} \leq b \cdot c^{-1}$
⟨*proof*⟩

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: $a \in G$ $b \in G$ **and** A2: $c \leq a \cdot b^{-1}$
shows $c \cdot b \leq a$
⟨*proof*⟩

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: $a \in G$ $b \in G$ **and** A2: $c \leq a^{-1} \cdot b$

```
  shows a·c≤ b
⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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
$c \cdot b^{-1} \leq a$
$a^{-1} \cdot c \leq b$
⟨*proof*⟩

If an element is greater or equal than another then the difference is nonnegative.

lemma (in group3) OrderedGroup_ZF_1_L9D: **assumes** A1: $a \leq b$
shows $\mathbf{1} \leq b \cdot a^{-1}$
⟨*proof*⟩

If an element is greater than another then the difference is positive.

lemma (in group3) OrderedGroup_ZF_1_L9E:
assumes A1: $a \leq b$ $a \neq b$
shows $\mathbf{1} \leq b \cdot a^{-1}$ $\mathbf{1} \neq b \cdot a^{-1}$ $b \cdot a^{-1} \in G_+$
⟨*proof*⟩

If the difference is nonnegative, then $a \leq b$.

lemma (in group3) OrderedGroup_ZF_1_L9F:
assumes A1: $a \in G$ $b \in G$ **and** A2: $\mathbf{1} \leq b \cdot a^{-1}$
shows $a \leq b$
⟨*proof*⟩

If we increase the middle term in a product, the whole product increases.

lemma (in group3) OrderedGroup_ZF_1_L10:
assumes $a \in G$ $b \in G$ **and** $c \leq d$
shows $a \cdot c \cdot b \leq a \cdot d \cdot b$
⟨*proof*⟩

A product of (strictly) positive elements is not the unit.

lemma (in group3) OrderedGroup_ZF_1_L11:
assumes A1: $\mathbf{1} \leq a$ $\mathbf{1} \leq b$
and A2: $\mathbf{1} \neq a$ $\mathbf{1} \neq b$
shows $\mathbf{1} \neq a \cdot b$
⟨*proof*⟩

A product of nonnegative elements is nonnegative.

lemma (in group3) OrderedGroup_ZF_1_L12:
assumes A1: $\mathbf{1} \leq a$ $\mathbf{1} \leq b$
shows $\mathbf{1} \leq a \cdot b$
⟨*proof*⟩

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: $a \leq b$ **shows** $\mathbf{1} \leq b \cdot a^{-1}$
⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

38.3 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

The G⁺ is G₊ plus the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L24: shows G⁺ = G₊∪{1}
  ⟨proof⟩
```

What is $-G_+$, really?

```
lemma (in group3) OrderedGroup_ZF_1_L25: shows
  (-G₊) = {a⁻¹. a∈G₊}
  (-G₊) ⊆ G
⟨proof⟩
```

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⟩
```

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₊
  ⟨proof⟩
```

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
```

$\mathbf{1} \notin G_+ \cup (-G_+)$
⟨proof⟩

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: $a \in G \quad b \in G$ **and** A2: $a \cdot b^{-1} \in G^+$
  shows $b \leq a$
⟨proof⟩

A special case of OrderedGroup_ZF_1_L28 when $a \cdot b^{-1}$ is positive.

corollary (in group3) OrderedGroup_ZF_1_L29:
  assumes A1: $a \in G \quad b \in G$ **and** A2: $a \cdot b^{-1} \in G_+$
  shows $b \leq a \quad b \neq a$
⟨proof⟩

A bit stronger that OrderedGroup_ZF_1_L29, adds case when two elements are equal.

lemma (in group3) OrderedGroup_ZF_1_L30:
  assumes $a \in G \quad b \in G$ **and** $a=b \vee b \cdot a^{-1} \in G_+$
  shows $a \leq b$
  ⟨proof⟩

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: $a \in G \quad b \in G$
  shows $a=b \vee (a \leq b \wedge a \neq b) \vee (b \leq a \wedge b \neq a)$
⟨proof⟩

38.4 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: $\forall g \in A.\ L \leq g \wedge g \leq M$
  and A2: S = RightTranslation(G,P,L^{-1})
  and A3: $a \in S(A)$
  shows $a \leq M \cdot L^{-1} \quad \mathbf{1} \leq a$
⟨proof⟩

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 $\exists B.\exists g \in G^+.\exists T \in G \rightarrow G.\ A = T(B) \wedge B \subseteq$ Interval(r,$\mathbf{1}$,g)

⟨*proof*⟩

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩

end
```

39 More on ordered groups

```
theory OrderedGroup_ZF_1 imports OrderedGroup_ZF

begin
```

In this theory we continue the `OrderedGroup_ZF` theory development.

39.1 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⟩
```

If $a \in G^+$, then $|a| = a$.

```
lemma (in group3) OrderedGroup_ZF_3_L2:
  assumes A1: a∈G⁺ shows |a| = a
⟨proof⟩
```

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 ⟨proof⟩
```

If a is positive, then $|a| = a$.

```
lemma (in group3) OrderedGroup_ZF_3_L2B:
  assumes a∈G₊ shows |a| = a
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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|
⟨*proof*⟩

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*⟩

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*⟩

$a^{-1} \leq |a|$ (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*⟩

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|
⟨*proof*⟩

$|a^{-1}| \leq |a|$.

lemma (**in** group3) OrderedGroup_ZF_3_L7:
assumes r {is total on} G **and** a∈G
shows |a^{-1}|≤|a|
⟨*proof*⟩

$|a^{-1}| = |a|$.

lemma (**in** group3) OrderedGroup_ZF_3_L7A:
assumes A1: r {is total on} G **and** A2: a∈G
shows |a^{-1}| = |a|
⟨*proof*⟩

$|a \cdot b^{-1}| = |b \cdot a^{-1}|$. 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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

$|a| \leq L$ 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

39.2 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)
  ⟨proof⟩
```

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⟩
```

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)|)
  ⟨proof⟩
```

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⟩
```

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⟩
```

39.3 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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**}
  ⟨*proof*⟩

39.4 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**⟩}
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

39.5 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⟩
```

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⟩
```

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 $\neq$ {1} **and**
  A3: X$\neq$0 **and** A4: f:G$\rightarrow$G **and**
  A5: $\forall$a$\in$G.$\exists$b$\in$G$_+$.$\forall$y. b$\leq$y $\longrightarrow$ f(y^{-1}) $\leq$ a **and**
  A6: $\forall$x$\in$X. b(x) $\in$ G $\wedge$ L $\leq$ f(b(x))
  shows $\exists$l.$\forall$x$\in$X. l $\leq$ b(x)
$\langle$*proof*$\rangle$

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 $\neq$ {1} **and**
  A3: X$\neq$0 **and** A4: f:G$\rightarrow$G **and**
  A5: $\forall$a$\in$G.$\exists$b$\in$G$_+$.$\forall$y. b$\leq$y $\longrightarrow$ a $\leq$ f(y) **and**
  A6: $\forall$a$\in$G.$\exists$b$\in$G$_+$.$\forall$y. b$\leq$y $\longrightarrow$ f(y^{-1}) $\leq$ a **and**
  A7: $\forall$x$\in$X. b(x) $\in$ G $\wedge$ L $\leq$ f(b(x)) $\wedge$ f(b(x)) $\leq$ U
shows $\exists$M.$\forall$x$\in$X. |b(x)| $\leq$ M
$\langle$*proof*$\rangle$

end

40 Rings - introduction

theory Ring_ZF **imports** AbelianGroup_ZF

begin

This theory file covers basic facts about rings.

40.1 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) $\equiv$ IsAgroup(R,A) $\wedge$ (A {is commutative on} R) $\wedge$

```
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
```

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
⟨proof⟩
```

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`
  ⟨*proof*⟩

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`
  ⟨*proof*⟩

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

Minus times minus is plus.

```
lemma (in ring0) Ring_ZF_1_L7A: assumes a∈R  b∈R
  shows (-a)·(-b) = a·b
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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 ⟨proof⟩
```

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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**
  ⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

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⟩
```

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
  ⟨proof⟩
```

Square of an element and its opposite are the same.

```
lemma (in ring0) Ring_ZF_1_L14:
  assumes a∈R shows (-a)² = ((a)²)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

The ring is trivial iff $0 = 1$.

```
lemma (in ring0) Ring_ZF_1_L17: shows R = {0} ⟷ 0=1
⟨proof⟩
```

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⟩
```

40.2 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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

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*⟩

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
  ⟨*proof*⟩

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*⟩

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)
  ⟨proof⟩
```

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⟩

end
```

41 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.

41.1 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 listsum (∑ _ 70)
  defines listsum_def [simp]: ∑s ≡ Fold(A,0,s)
```

```
  fixes listprod (∏ _ 70)
  defines listprod_def [simp]: ∏s ≡ Fold(M,1,s)

  fixes nat_mult (infix · 95)
  defines nat_mult_def [simp]: n·x ≡ ∑{⟨k,x⟩. k∈n}

  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 ring3 < add_monoid: monoid1 R A ringa ringzero listsum nat_mult
  ⟨proof⟩
```

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.

```
sublocale ring3 < mul_monoid: monoid1 R M ringm ringone listprod pow
  ⟨proof⟩
```

$0 \cdot x = 0$ 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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

The usual properties of multiples and powers: $(n+1)x = nx+x$ and $x^n+1 = x^n x$. 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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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})
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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)
```

⟨*proof*⟩

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)
  ⟨proof⟩
```

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⟩
```

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⟩

end
```

42 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.

42.1 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*⟩

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*⟩

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*⟩

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*⟩

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)
  ⟨*proof*⟩

end

43 Ordered rings

theory OrderedRing_ZF **imports** Ring_ZF OrderedGroup_ZF_1

begin

In this theory file we consider ordered rings.

43.1 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)
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

The order relation in rings is transitive.

lemma (**in** ring1) ring_ord_transitive: **assumes** A1: a≤b b≤c
  shows a≤c
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

Taking minus on both sides reverses an inequality.

lemma (**in** ring1) OrdRing_ZF_1_L4B: **assumes** a≤b
  shows (-b) ≤ (-a)
  ⟨*proof*⟩

The next lemma just expands the condition that requires the set of nonnegative 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
  ⟨*proof*⟩

Double nonnegative is nonnegative.

lemma (**in** ring1) OrdRing_ZF_1_L5A: **assumes** A1: **0**≤a
  shows **0**≤**2**·a
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

$a \leq b$ 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**
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

A square is nonnegative.

lemma (in ring1) OrdRing_ZF_1_L10:
assumes A1: a∈R **shows** **0**≤(a^2)
⟨*proof*⟩

1 is nonnegative.

corollary (in ring1) ordring_one_is_nonneg: **shows** **0** ≤ **1**
⟨*proof*⟩

In nontrivial rings one is positive.

lemma (in ring1) ordring_one_is_pos: **assumes** **0**≠**1**
shows **1** ∈ R_+
⟨*proof*⟩

Nonnegative is not negative. Property of ordered groups.

lemma (in ring1) OrdRing_ZF_1_L11: **assumes** **0**≤a
shows ¬(a≤**0** ∧ a≠**0**)
⟨*proof*⟩

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⟩
```

If $a \leq b$, then $0 \leq b - a$.

```
lemma (in ring1) OrdRing_ZF_1_L13: assumes a≤b
  shows 0 ≤ b-a
  ⟨proof⟩
```

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+
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

43.2 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|
  ⟨proof⟩
```

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|
  ⟨proof⟩
```

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|
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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|
  ⟨proof⟩
```

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
  ⟨proof⟩
```

43.3 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
  ⟨proof⟩
```

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+)
  ⟨proof⟩
```

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+
  ⟨proof⟩
```

`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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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
  ⟨*proof*⟩

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**
  ⟨*proof*⟩

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*⟩

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*⟩

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⟩
```

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+
  ⟨proof⟩
```

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+
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

end

44 Cardinal numbers

theory `Cardinal_ZF` **imports** `ZF.CardinalArith 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.

44.1 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⟩
```

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⟩
```

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*⟩

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*⟩

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`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

44.2 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*⟩

44.3 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 files to be included in a file.

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 finite choice always holds.

theorem `finite_choice:`
  assumes `n∈nat`
  shows `{the axiom of} n {choice holds}`
⟨*proof*⟩

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)`

⟨*proof*⟩

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})` ⟨*proof*⟩

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*⟩

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*⟩

end

45 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.

45.1 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*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

The trivial subgroup is then a normal subgroup.

```
corollary (in group0) trivial_normal_subgroup:
  shows IsAnormalSubgroup(G,P,{1})
⟨proof⟩
```

The whole group is normal as a subgroup

```
lemma (in group0) whole_normal_subgroup:
  shows IsAnormalSubgroup(G,P,G)
⟨proof⟩
```

45.2 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*⟩

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*⟩

45.3 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)
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

45.4 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 ($\langle\_\rangle_G$ 80)
  where X⊆G $\Longrightarrow$ $\langle X\rangle_G$ ≡ ⋂{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($\langle X\rangle_G$,P)
⟨*proof*⟩

The generated subgroup contains the original set

theorem (**in** group0) subgroupGen_contains_set:
  assumes X⊆G
  shows X ⊆ $\langle X\rangle_G$
⟨*proof*⟩

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 $\langle X\rangle_G$ ⊆ H
⟨*proof*⟩

end

46 Groups 5

theory Group_ZF_5 **imports** Group_ZF_4 Ring_ZF Semigroup_ZF

begin

In this theory we study group homomorphisms.

46.1 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 is 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)⟩
⟨proof⟩
```

An endomorphism is a homomorphism from a group to the same group. In case the group is abelian, it has a nice structure.

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)
⟨proof⟩
```

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)
⟨proof⟩
```

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⟩
```

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, with 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⟩
```

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) F = P {lifted to function space over}
G
  shows F⟨f,g⟩ ∈ End(G,P)
⟨proof⟩
```

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))
  ⟨proof⟩
```

The inverse of an abelian group is an endomorphism.

```
lemma (in abelian_group) end_inverse_group:
  shows GroupInv(G,P) ∈ End(G,P)
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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))
  ⟨proof⟩
```

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⟩
  ⟨proof⟩
```

46.2 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)})
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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))
  ⟨proof⟩
```

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
  ⟨proof⟩

end
```

47 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.

47.1 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
  ⟨proof⟩
```

The singleton containing zero is and ideal.

```
lemma (in ring0) zero_ideal:
  shows {0} ◁R ⟨proof⟩
```

An ideal is s subset of the the ring.

```
lemma (in ring0) ideal_dest_subset:
  assumes I ◁R
  shows I ⊆ R ⟨proof⟩
```

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 ⟨proof⟩
```

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 ⟨proof⟩
```

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
  ⟨proof⟩
```

Every ideals contains zero.

```
lemma (in ring0) ideal_dest_zero:
  assumes I ◁R
  shows 0 ∈ I
  ⟨proof⟩
```

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⟩
```

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 ∀J∈𝒥. (J ◁R) 𝒥 ≠ 0
  shows (⋂𝒥)  ◁R
  ⟨proof⟩
```

In particular, intersection of two ideals is an ideal.

```
corollary (in ring0) inter_two_ideals: assumes I◁R J◁R
  shows (I∩J) ◁R
⟨proof⟩
```

From any set, we may construct the minimal ideal containing that set

```
definition (in ring0) generatedIdeal (⟨_⟩_I)
  where X⊆R ⟹ ⟨X⟩_I ≡ ⋂{I∈ℐ. X ⊆ I}
```

The ideal generated by a set is an ideal

```
corollary (in ring0) generated_ideal_is_ideal:
  assumes X⊆R shows ⟨X⟩_I ◁R
⟨proof⟩
```

The ideal generated by a set is contained in any ideal containing the set.

```
corollary (in ring0) generated_ideal_small:
  assumes X⊆I I ◁R
  shows ⟨X⟩_I ⊆ I
⟨proof⟩
```

The ideal generated by a set contains the set.

```
corollary (in ring0) generated_ideal_contains_set:
  assumes X⊆R shows X⊆⟨X⟩_I
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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 (_◁_m R) where
  I◁_m R ≡ 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·_I J ≡ ⟨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+_I J ≡ ⟨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 ⟹ ⊕_I 𝒥 ≡ ⟨⋃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+_I J and J ⊆ I+_I J and I∪J ⊆ I+_I J
⟨proof⟩
```

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 $\mathcal{J} \subseteq \mathcal{I}$ $s\in(\oplus_I\mathcal{J})$
  shows $\exists\mathcal{T}\in\mathtt{FinPow}(\mathcal{J}).\ s\in(\oplus_I\mathcal{T})$
⟨*proof*⟩

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 $I\triangleleft R$ $J\triangleleft R$
  shows $I\cdot_I J \subseteq I\cap J$ **and** $(I\cdot_I J)\triangleleft R$ **and** $M(I\times J) \subseteq I\cdot_I J$
⟨*proof*⟩

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 $I \triangleleft R$ $J \triangleleft R$ $x\in I$ $y\in J$
  shows $x+y \in I+_I J$
⟨*proof*⟩

For two ideals the set containing all sums of their elements is also an ideal.

lemma (**in** ring0) sum_elements_is_ideal:
  assumes $I \triangleleft R$ $J \triangleleft R$
  shows $(A(I\times J)) \triangleleft R$
⟨*proof*⟩

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 $I \triangleleft R$ $J \triangleleft R$
  shows $(A(I \times J)) = I+_I J$
⟨*proof*⟩

The sum ideal of two ideals is indeed an ideal.

corollary (**in** ring0) sum_ideals_is_ideal:
  assumes $I \triangleleft R$ $J \triangleleft R$
  shows $(I+_I J) \triangleleft R$ ⟨*proof*⟩

The operation of taking the sum of ideals is commutative.

corollary (**in** ring0) sum_ideals_commute:
  assumes $I\triangleleft R$ $J\triangleleft R$
  shows $(I +_I J) = (J +_I I)$
⟨*proof*⟩

Now that we know what the product of ideals is, we are able to define what a prime ideal is:

definition (**in** ring0) primeIdeal ($\_\triangleleft_p R$) **where**
  $P\triangleleft_p R \equiv P\triangleleft R \wedge P\neq R \wedge (\forall I\in\mathcal{I}.\ \forall J\in\mathcal{I}.\ I\cdot_I J \subseteq P \longrightarrow (I\subseteq P \vee J\subseteq P))$

Any maximal ideal is a prime ideal.

```
theorem (in ring0) maximal_is_prime:
  assumes Q◁ₘR shows Q◁ₚR
⟨proof⟩
```

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} ◁ₚR
⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

47.2 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)
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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**
  $\mathbf{0}_I$ ≡ 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**
  $\mathbf{1}_I$ ≡ 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**
  $\mathbf{2}_I$ ≡ 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))
  ⟨*proof*⟩

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))
  ⟨proof⟩
```

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}⟩
  ⟨proof⟩
```

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}⟩
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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). (⊕_I𝒥) = (⊕_I𝒯)
⟨proof⟩

end
```

48 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.

48.1 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+_Ry ≡ A⟨x,y⟩

  fixes ringminus (-_R _ 89)
```

```
  defines ringminus_def [simp]: (-Rx) ≡ GroupInv(R,A)(x)

  fixes ringsub (infixl -R 90)
  defines ringsub_def [simp]: x-Ry ≡ x+R(-Ry)

  fixes ringm (infixl ·R 95)
  defines ringm_def [simp]: x·Ry ≡ M⟨x,y⟩

  fixes ringzero (0R)
  defines ringzero_def [simp]: 0R ≡ TheNeutralElement(R,A)

  fixes ringone (1R)
  defines ringone_def [simp]: 1R ≡ TheNeutralElement(R,M)

  fixes ringtwo (2R)
  defines ringtwo_def [simp]: 2R ≡ 1R+R1R

  fixes ringsq (_^2R [96] 97)
  defines ringsq_def [simp]: x^2R ≡ x·Rx

  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
```

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)
```

$I \lhd R_t$ means that I is an ideal of S.

abbreviation (**in** ring_homo) ideal_target (_◁R$_t$)
  where I◁R$_t$ ≡ ring0.Ideal(S,U,V,I)

$I \lhd_p R_o$ means that I is a prime ideal of R.

abbreviation (**in** ring_homo) prime_ideal_origin (_◁$_p$R$_o$)
  where I◁$_p$R$_o$ ≡ ring0.primeIdeal(R,A,M,I)

We will write $I \lhd_p R_t$ to denote that I is a prime ideal of the ring S.

abbreviation (**in** ring_homo) prime_ideal_target (_◁$_p$R$_t$)
  where I◁$_p$R$_t$ ≡ 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-{$\mathbf{0}_S$}

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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) = $\mathbf{0}_S$}
⟨*proof*⟩

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) = $\mathbf{0}_S$
⟨*proof*⟩

As a ring homomorphism f preserves multiplication.

lemma (**in** ring_homo) homomor_dest_mult:
assumes x∈R y∈R
shows f(x$\cdot_R$y) = (f(x))$\cdot_S$(f(y))
⟨*proof*⟩

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))
⟨*proof*⟩

For $x \in R$ the value of f is in S.

lemma (**in** ring_homo) homomor_val: **assumes** x∈R
shows f(x) ∈ S
⟨*proof*⟩

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))
⟨*proof*⟩

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))
⟨*proof*⟩

A ring homomorphism maps zero to zero.

lemma (**in** ring_homo) homomor_dest_zero:
shows f($\mathbf{0}_R$) = $\mathbf{0}_S$
⟨*proof*⟩

The kernel of a homomorphism is never empty.

lemma (**in** `ring_homo`) `kernel_non_empty`:
  shows $\mathbf{0}_R$ ∈`ker` **and** `ker`≠`0`
  ⟨*proof*⟩

The image of the kernel by f is the singleton $\{0_R\}$.

corollary (**in** `ring_homo`) `image_kernel_2`: **shows** `f(ker) = {`$\mathbf{0}_S$`}`
⟨*proof*⟩

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*⟩

Kernel of the homomorphism in an ideal.

lemma (**in** `ring_homo`) `kernel_ideal`: **shows** `ker` ◁R_o
  ⟨*proof*⟩

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`$\triangleleft_p$$\mathrm{R}_t$ **shows** `f-(J) ≠ R`
⟨*proof*⟩

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`$\triangleleft_p$$\mathrm{R}_t$ `V {is commutative on} S`
  shows `(f-(J))`$\triangleleft_p$$\mathrm{R}_o$
⟨*proof*⟩

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`$\triangleleft_p$$\mathrm{R}_t$ `f ∈ surj(R,S)`
  shows `(f-(J))`$\triangleleft_p$$\mathrm{R}_o$
⟨*proof*⟩

48.2 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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

The quotient map is surjective

```
lemma (in ring2) quot_fun:
  shows f_I∈surj(R,R_I)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

The ideal we divide by is the kernel of the quotient map.

```
lemma (in ring2) quotient_kernel:
  shows quot_homomorphism.kernel = I
⟨proof⟩
```

The theorems proven in the `ring0` context are valid in the `ring 2` context when applied to the quotient ring.

```
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)
  ⟨proof⟩
```

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⟩
```

48.3 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*⟩

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*⟩

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*⟩

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◁R_t K◁R_t f∈surj(R,S)
  shows target_ring.productIdeal(J, K) =
    f(origin_ring.productIdeal((f-(J)), (f-(K))))
⟨*proof*⟩

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◁$_p$$R_o$ ker ⊆ J f∈surj(R,S)
  shows (f(J))◁$_p$$R_t$
⟨*proof*⟩

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∈$\mathcal{I}$. ker ⊆ K})
⟨*proof*⟩

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*⟩

Since the map is surjective, this bijection restricts to prime ideals on both sides.

corollary (**in** ring_homo) prime_ideal_quot_3:
  assumes K◁R_t f ∈ surj(R,S)
  shows K◁$_p$$R_t$ ⟷ ((f-(K))◁$_p$R)
⟨*proof*⟩

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◁$_p$$R_t$}) ∈
    bij({J∈Pow(S). J◁$_p$$R_t$}, {J∈Pow(R). ker⊆J ∧ (J◁$_p$R)})
⟨*proof*⟩

end

49 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*⟩

theorem (**in** commutative_ring) principal_ideal:
  assumes x∈R
  shows ⟨{x}⟩$_I$ = {x · y . y ∈ R}
⟨*proof*⟩

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) ⟨proof⟩

lemma (in commutative_ring) prime_ring_zero_divs_2:
  assumes HasNoZeroDivs(R,A,M)
  shows [R,A,M]{is a prime ring} ⟨proof⟩

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⟩


end
```

50 Fields - introduction

```
theory Field_ZF imports Ring_ZF

begin
```

This theory covers basic facts about fields.

50.1 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: $\mathbf{0} \neq \mathbf{1}$

assumes inv_exists: ∀x∈K. x≠**0** ⟶ (∃y∈K. x·y = **1**)

fixes non_zero (K_0)
defines non_zero_def[simp]: K_0 ≡ K-{**0**}

fixes inv (_$^{-1}$ [96] 97)
defines inv_def[simp]: a^{-1} ≡ GroupInv(K_0,restrict(M,K_0×K_0))(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)
⟨*proof*⟩

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)
⟨*proof*⟩

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
⟨*proof*⟩

Fields do not have zero divisors.

lemma (in field0) field_has_no_zero_divs: **shows** HasNoZeroDivs(K,A,M)
⟨*proof*⟩

K_0 (the set of nonzero field elements is closed with respect to multiplication.

lemma (in field0) Field_ZF_1_L2:
shows K_0 {is closed under} M
⟨*proof*⟩

Any nonzero element has a right inverse that is nonzero.

lemma (in field0) Field_ZF_1_L3: **assumes** A1: a∈K_0
shows ∃b∈K_0. a·b = **1**
⟨*proof*⟩

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(K_0,restrict(M,K_0×K_0))
group0(K_0,restrict(M,K_0×K_0))
1 = TheNeutralElement(K_0,restrict(M,K_0×K_0))

⟨*proof*⟩

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*⟩

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*⟩

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
```
⟨*proof*⟩

50.2 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*⟩

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
```
⟨*proof*⟩

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*⟩

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
```

⟨*proof*⟩

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
  ⟨proof⟩
```

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⟩
```

50.3 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⟩
```

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⟩

end
```

51 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.

51.1 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 the domain of 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 $\mathcal{M} \to \mathcal{M}$ 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)` $\equiv$ `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)` $\equiv$ `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))`
  ⟨*proof*⟩

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, V is the group, A_V is the group operation, and H is the ring homomorphism that of the ring of scalars to the ring of endomorphisms of the group. On the right hand side of the definition `End(V,`A_V`)` is the set of endomorphisms, 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)` $\equiv$ `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)` $\equiv$
  `IsAring(S,A,M)` $\wedge$ `IsAgroup(`$\mathcal{M}$`,`A_M`)` $\wedge$ `(`A_M `{is commutative on}` $\mathcal{M}$`)` $\wedge$
`IsAction(S,A,M,`$\mathcal{M}$`,`A_M`,H)`

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)`
  ⟨*proof*⟩

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`)`
  ⟨*proof*⟩

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`

⟨*proof*⟩

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)
  ⟨proof⟩
```

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⟩
  ⟨proof⟩
```

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⟩
```

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
```

⟨*proof*⟩

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** $\Theta \in \mathcal{M}$
⟨*proof*⟩

Θ is indeed the neutral element of addition of module elements.

lemma **(in** `module0`) `zero_neutral`: **assumes** x∈$\mathcal{M}$
shows x $+_V$ Θ = x **and** Θ $+_V$ x = x
⟨*proof*⟩

51.2 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 $\cdot_S$ Θ = Θ
⟨*proof*⟩

The scalar multiplication is distributive with respect to the module addition.

lemma **(in** `module0`) `module_ax1`: **assumes** r∈R x∈$\mathcal{M}$ y∈$\mathcal{M}$
shows r$\cdot_S$(x$+_V$y) = r$\cdot_S$x $+_V$ r$\cdot_S$y
⟨*proof*⟩

The scalar addition is distributive with respect to scalar multiplication.

lemma **(in** `module0`) `module_ax2`: **assumes** r∈R s∈R x∈$\mathcal{M}$
shows (r+s)$\cdot_S$x = r$\cdot_S$x $+_V$ s$\cdot_S$x
⟨*proof*⟩

Multiplication by scalars is associative with multiplication of scalars.

lemma **(in** `module0`) `module_ax3`: **assumes** r∈R s∈R x∈$\mathcal{M}$
shows (r·s)$\cdot_S$x = r$\cdot_S$(s$\cdot_S$x)
⟨*proof*⟩

Scaling a module element by one gives the same module element.

lemma **(in** `module0`) `module_ax4`: **assumes** x∈$\mathcal{M}$ **shows** **1**$\cdot_S$x = x
⟨*proof*⟩

Multiplying by zero is zero.

lemma **(in** `module0`) `mult_zero`:

assumes g∈$\mathcal{M}$ **shows** $\mathbf{0}\cdot_S$g = Θ
⟨*proof*⟩

Taking inverses in a module is just multiplying by -1

lemma (in module0) inv_module:
assumes g∈$\mathcal{M}$
shows (-**1**)$\cdot_S$g = − g
⟨*proof*⟩

end

51.3 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→$\mathcal{M}$ ⟹ D∈FinPow(C) ⟹ LinearComb(D,fR,fG)
≡ if D≠0 then CommSetFold(A_M,{⟨m,(fRm)$\cdot_S$ (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→$\mathcal{M}$
shows {⟨m,(AAm)$\cdot_S$ (Bm)⟩. m∈C}:C→$\mathcal{M}$
⟨*proof*⟩

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→$\mathcal{M}$ D∈FinPow(C)
shows (∑ [D;{AA,B}])∈$\mathcal{M}$
⟨*proof*⟩

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→$\mathcal{M}$
shows ∑ [{x};{AA,B}]=(AAx)$\cdot_S$(Bx)

⟨proof⟩

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→$\mathcal{M}$ 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⟩

A linear combination can always be defined on a cardinal.

lemma(in module0) linComb_reorder_terms1:
  assumes AA:X→R B:X→$\mathcal{M}$ D∈FinPow(X) g∈bij(|D|,D)
  shows (∑ [D;{AA,B}])=∑ [|D|;{AA O g,B O g}]
⟨proof⟩

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→$\mathcal{M}$ D∈FinPow(X) g∈bij(E,D)
  shows (∑ [D;{AA,B}])=∑ [E;{AA O g,B O g}]
⟨proof⟩

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→$\mathcal{M}$ D∈FinPow(X)
  shows (∑ [D;{AA,B}])=∑ [D;{restrict(AA,D),restrict(B,D)}]
  ⟨proof⟩

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→$\mathcal{M}$ D∈FinPow(X)
  shows ∃n∈nat. ∃A1∈n→R. ∃B1∈n→$\mathcal{M}$. ∑ [D;{AA,B}]=∑ [n;{A1,B1}] ∧ A1n=AAD
∧ B1n=BD
⟨proof⟩

51.3.1 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→$\mathcal{M}$ D∈FinPow(X) AA1:Y→R B1:Y→$\mathcal{M}$
  shows (∑ [D;{AA,B}])+$_V$(∑ [0;{AA1,B1}])=∑ [D;{AA,B}]
⟨proof⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

51.3.2 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 $\Theta \in$T T$\subseteq \mathcal{M}$ R$\neq${**0**}
  shows $\neg$(T{is linearly independent})
⟨*proof*⟩

51.4 Submodule

A submodule is a subgroup that is invariant by the action

definition(**in** module0)
  IsAsubmodule
  where IsAsubmodule($\mathcal{N}$) $\equiv$ ($\forall$r$\in$R. $\forall$h$\in\mathcal{N}$. r$\cdot_S$ h$\in\mathcal{N}$) $\wedge$ IsAsubgroup($\mathcal{N}$,A_M)

lemma(**in** module0) sumodule_is_subgroup:
  assumes IsAsubmodule($\mathcal{N}$)
  shows IsAsubgroup($\mathcal{N}$,A_M)
  ⟨*proof*⟩

lemma(**in** module0) sumodule_is_subaction:
  assumes IsAsubmodule($\mathcal{N}$) r$\in$R h$\in\mathcal{N}$
  shows r$\cdot_S$ h$\in\mathcal{N}$
  ⟨*proof*⟩

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 $\forall$r$\in$R. $\forall$h$\in\mathcal{N}$. r$\cdot_S$ h$\in\mathcal{N}$ $\mathcal{N}\subseteq\mathcal{M}$
  shows $\forall$h$\in\mathcal{N}$. ($-$h)$\in\mathcal{N}$
⟨*proof*⟩

corollary(**in** module0) submoduleI:
  assumes $\mathcal{N}\subseteq\mathcal{M}$ $\mathcal{N}\neq$0 $\mathcal{N}${is closed under}A_M $\forall$r$\in$R. $\forall$h$\in\mathcal{N}$. r$\cdot_S$ h$\in\mathcal{N}$
  shows IsAsubmodule($\mathcal{N}$) ⟨*proof*⟩

Every module has at least two submodules: the whole module and the trivial module.

corollary(**in** module0) trivial_submodules:
  shows IsAsubmodule($\mathcal{M}$) **and** IsAsubmodule({Θ})
  ⟨*proof*⟩

The restriction of the action is an action.

lemma(**in** module0) action_submodule:
  assumes IsAsubmodule($\mathcal{N}$)
  shows {⟨r,restrict(Hr,$\mathcal{N}$)⟩. r$\in$R}:R$\rightarrow$End($\mathcal{N}$,restrict(A_M,$\mathcal{N}\times\mathcal{N}$))
⟨*proof*⟩

A submodule is a module with the restricted action.

corollary(**in** module0) submodule:
  assumes IsAsubmodule($\mathcal{N}$)
  shows IsLeftModule(R,A,M,$\mathcal{N}$,restrict(A_M,$\mathcal{N}\times\mathcal{N}$),{⟨r,restrict(Hr,$\mathcal{N}$)⟩. r∈R})
  ⟨*proof*⟩

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($\mathcal{N}$) D∈FinPow(X) AA:X→R B:X→$\mathcal{N}$
  shows ∑[D;{AA,B}]∈$\mathcal{N}$
⟨*proof*⟩

51.4.1 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∈$\mathcal{M}$→R. ∑[D;{AAA,BB}]=∑[BBD;{AA,id($\mathcal{M}$)}] ∧ (∀x∈$\mathcal{M}$-BBD. AAx=**0**)
⟨*proof*⟩

A span over a set is the collection over all linear combinations on those elements.

definition(**in** module0)
  Span({span of}_)
  where T⊆$\mathcal{M}$ ⟹ {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⊆$\mathcal{M}$
  shows IsAsubmodule({span of}T)
  and T⊆{span of}T
⟨*proof*⟩

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→$\mathcal{M}$ D∈FinPow(X)

```
  shows ∑[D;{AA,B}]∈({span of}(BD))
⟨proof⟩
```

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⟩

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.

51.5 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) ⟨proof⟩
```

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⟩
```

The previous map respects multiplication because of associativity

```
lemma (in ring0) action_regular_assoc:
  assumes g₁ ∈ R g₂ ∈ R
  shows {⟨xa, M ⟨(M ⟨g₁, g₂⟩), xa⟩⟩ . xa ∈ R} =
       EndMult(R, A) ⟨{⟨xa, M ⟨g₁, xa⟩⟩ . xa ∈ R}, {⟨xa, M ⟨g₂, xa⟩⟩
. xa ∈ R} ⟩
⟨proof⟩
```

The previous map takes the unit element to the identity map

```
lemma (in ring0) action_regular_neut:
  shows {⟨x, {⟨xa, M  ⟨x, xa⟩⟩ . xa ∈ R}⟩ . x ∈ R}  1 = id(R)
⟨proof⟩
```

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}) ⟨proof⟩
```

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}) ⟨proof⟩
```

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⟩
```

51.6 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 ⟨proof⟩
```

Annihilator is reverse monotonic

```
lemma (in module0) ann_mono:
  assumes N ⊆ 𝓜 K ⊆ N
  shows ann(N) ⊆ ann(K)
⟨proof⟩
```

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⟩
```

Annihilators on commutative rings are ideals

```
corollary (in module0) comm_ann_ideal:
```

```
  assumes N ⊆ ℳ M {is commutative on} R
  shows ann(N) ◁R ⟨proof⟩
```

end

52 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.

52.1 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 $(s_1 + s_2)v = s_1v + s_2v$ 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.

```
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]: $v_1 +_V v_2 \equiv A_V\langle v_1,v_2\rangle$

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(V,A_V)(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 vector spaces in the `vector_space0` context.

lemma (**in** vector_space0) V_vec_space: **shows** IsVectorSpace(K,A,M,V,A_V,H)
⟨*proof*⟩

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,A_V,H)
shows vector_space0(K, A, M, V, A_V, H)
⟨*proof*⟩

The assumptions of `module0` context hold in the `vector_spce0` context.

lemma (**in** vector_space0) vec_spce_mod: **shows** module0(K, A, M, V, A_V, H)
⟨*proof*⟩

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 V A_V
⟨*proof*⟩

52.2 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
  ⟨*proof*⟩

Vector addition is commutative.

lemma (**in** vector_space0) vec_spce_ax2: **assumes** u∈V v∈V
  shows u $+_V$ v = v $+_V$ u
  ⟨*proof*⟩

The zero vector is a vector.

lemma (**in** vector_space0) vec_spce_ax3a: **shows** Θ ∈ V
  ⟨*proof*⟩

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
  ⟨*proof*⟩

The additive inverse of a vector is a vector.

lemma (**in** vector_space0) vec_spce_ax4a: **assumes** v∈V **shows** (−v) ∈ V
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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$\cdot_S$(y$\cdot_S$v) = (x·y)$\cdot_S$v
  ⟨*proof*⟩

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**$\cdot_S$v = v
  ⟨*proof*⟩

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$\cdot_S$(u$+_V$v) = x$\cdot_S$u $+_V$ x$\cdot_S$v
  ⟨*proof*⟩

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
  ⟨*proof*⟩

end

53 Ordered fields

theory `OrderedField_ZF` **imports** `OrderedRing_ZF Field_ZF`

begin

This theory covers basic facts about ordered fiels.

53.1 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)
⟨*proof*⟩

Ordered field is a field, of course.

lemma OrdField_ZF_1_L1A: **assumes** IsAnOrdField(K,A,M,r)
shows IsAfield(K,A,M)
⟨*proof*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

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: $\forall a\in R_+.\ \exists b\in R.\ a\cdot b = \mathbf{1}$ **and** A2: $c\in R$ $c\neq \mathbf{0}$
shows $\exists b\in R.\ c\cdot b = \mathbf{1}$
⟨*proof*⟩

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 $\mathbf{0} \neq \mathbf{1}$ **and** M {is commutative on} R
and $\forall a\in R_+.\ \exists b\in R.\ a\cdot b = \mathbf{1}$
shows IsAnOrdField(R,A,M,r)
⟨*proof*⟩

The set of positive field elements is closed under multiplication.

lemma **(in** field1**)** OrdField_ZF_1_L5: **shows** R_+ {is closed under} M
⟨*proof*⟩

The set of positive field elements is closed under multiplication: the explicit version.

lemma **(in** field1**)** pos_mul_closed:
assumes A1: $\mathbf{0} < a$ $\mathbf{0} < b$
shows $\mathbf{0} < a\cdot b$
⟨*proof*⟩

In fields square of a nonzero element is positive.

lemma **(in** field1**)** OrdField_ZF_1_L6: **assumes** $a\in R$ $a\neq \mathbf{0}$

```
  shows a² ∈ R₊
  ⟨proof⟩
```

The next lemma restates the fact `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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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₊
  ⟨proof⟩
```

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 exists the (multiplicative) inverse of a^2+b^2.

```
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⟩
```

53.2 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
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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
   ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

Sometimes we can reverse an inequality by taking inverse on both sides.

lemma (**in** field1) OrdField_ZF_2_L7:
  assumes A1: $a \in R_+$ **and** A2: $a^{-1} \leq b$
  shows $b^{-1} \leq a$
⟨*proof*⟩

Sometimes we can reverse a strict inequality by taking inverse on both sides.

lemma (**in** field1) OrdField_ZF_2_L8:
  assumes A1: $a \in R_+$ **and** A2: $a^{-1} < b$
  shows $b^{-1} < a$
⟨*proof*⟩

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*⟩

53.3 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

54 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.

54.1 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 prependig + 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 $\leq$ 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*⟩

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 (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 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))
```

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⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

Integers with addition and integers with multiplication form monoids.

theorem (**in** int0) Int_ZF_1_T1: **shows**
  IsAmonoid(ℤ,IntegerAddition)
  IsAmonoid(ℤ,IntegerMultiplication)
⟨*proof*⟩

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*⟩

0 and 1, as defined in int0 context, are integers.

lemma (**in** int0) Int_ZF_1_L8A: **shows** **0** ∈ ℤ **1** ∈ ℤ
⟨*proof*⟩

Zero is not one.

lemma (**in** int0) int_zero_not_one: **shows** **0** ≠ **1**
⟨*proof*⟩

The set of integers is not empty, of course.

lemma (**in** int0) int_not_empty: **shows** ℤ ≠ 0
  ⟨*proof*⟩

The set of integers has more than just zero in it.

lemma (**in** int0) int_not_trivial: **shows** ℤ ≠ {**0**}
  ⟨*proof*⟩

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*⟩

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)
  ⟨*proof*⟩

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*⟩

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
  ⟨proof⟩
```

Negative of zero is zero.

```
lemma (in int0) Int_ZF_1_L11: shows (-0) = 0
  ⟨proof⟩
```

A trivial calculation lemma that allows to subtract and add one.

```
lemma Int_ZF_1_L12:
  assumes m∈int shows m $- $#1 $+ $#1 = m
  ⟨proof⟩
```

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
  ⟨proof⟩
```

Adding or subtracing one changes integers.

```
lemma (in int0) Int_ZF_1_L14: assumes A1: m∈ℤ
  shows
  m+1 ≠ m
  m-1 ≠ m
⟨proof⟩
```

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⟩
```

54.2 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⟩
```

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⟩
```

Integer order is a relation on integers.

lemma `Int_ZF_2_L1B`: **shows** `IntegerOrder` $\subseteq$ `int`$\times$`int`
⟨*proof*⟩

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`$\subseteq\mathbb{Z}$
⟨*proof*⟩

The order on integers is reflexive.

lemma **(in** `int0`**)** `int_ord_is_refl`: **shows** `refl(`$\mathbb{Z}$`,IntegerOrder)`
  ⟨*proof*⟩

The essential condition to show antisymmetry of the order on integers.

lemma **(in** `int0`**)** `Int_ZF_2_L3`:
  assumes `A1`: $m \leq n$ $n \leq m$
  shows `m=n`
⟨*proof*⟩

The order on integers is antisymmetric.

lemma **(in** `int0`**)** `Int_ZF_2_L4`: **shows** `antisym(IntegerOrder)`
⟨*proof*⟩

The essential condition to show that the order on integers is transitive.

lemma `Int_ZF_2_L5`:
  assumes `A1`: $\langle m,n\rangle \in$ `IntegerOrder` $\langle n,k\rangle \in$ `IntegerOrder`
  shows $\langle m,k\rangle \in$ `IntegerOrder`
⟨*proof*⟩

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`: $m\leq n$ $n\leq k$
  shows $m\leq k$
⟨*proof*⟩

The order on integers is transitive.

lemma `Int_ZF_2_L6`: **shows** `trans(IntegerOrder)`
⟨*proof*⟩

The order on integers is a partial order.

lemma `Int_ZF_2_L7`: **shows** `IsPartOrder(int,IntegerOrder)`
  ⟨*proof*⟩

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⟩
```

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⟩
```

If a pair (i, m) belongs to the order relation on integers and $i \neq m$, then $i < m$ 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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
⟨*proof*⟩

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)
⟨*proof*⟩

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
⟨*proof*⟩

If an integer is nonpositive, then its opposite is nonnegative.

lemma (in int0) Int_ZF_2_L10A: **assumes** k ≤ **0**
shows **0**≤(-k)
⟨*proof*⟩

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**
⟨*proof*⟩

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*⟩

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*⟩

Adding one increases integers.

lemma (in int0) Int_ZF_2_L12A: **assumes** A1: j≤k
shows j ≤ k $+ $#1 j ≤ k+**1**
⟨*proof*⟩

Adding one increases integers, yet one more version.

lemma (in int0) Int_ZF_2_L12B: **assumes** A1: m∈ℤ **shows** m ≤ m+**1**

⟨*proof*⟩

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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 $\leq$ m+L+n
⟨*proof*⟩

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
⟨*proof*⟩

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)
⟨*proof*⟩

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
⟨*proof*⟩

Triangle inequality for integers.

lemma (**in** int0) Int_triangle_ineq:
assumes m∈**ℤ** n∈**ℤ**
shows abs(m+n)≤abs(m)+abs(n)
⟨*proof*⟩

Taking absolute value does not change nonnegative integers.

lemma (**in** int0) Int_ZF_2_L16:
assumes **0**≤m **shows** m∈$\mathbb{Z}^+$ **and** abs(m) = m
⟨*proof*⟩

$0 \leq 1$, so $|1| = 1$.

lemma (**in** int0) Int_ZF_2_L16A: **shows** **0**≤**1** **and** abs(**1**) = **1**
⟨*proof*⟩

$1 \leq 2$.

lemma (**in** int0) Int_ZF_2_L16B: **shows** **1**≤**2**
⟨*proof*⟩

Integers greater or equal one are greater or equal zero.

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⟩
```

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)
  ⟨proof⟩
```

The absolute value of zero is zero.

```
lemma (in int0) Int_ZF_2_L18: shows abs(0) = 0
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

$|m - n| = |n - m|$.

```
lemma (in int0) Int_ZF_2_L20: assumes m∈ℤ  n∈ℤ
  shows abs(m-n) = abs(n-m)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

Absolute value is nonnegative.

```
lemma (in int0) int_abs_nonneg: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ⁺  0 ≤ abs(m)
⟨proof⟩
```

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
  ⟨proof⟩
```

54.3 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

54.4 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*⟩

A trivial calculation lemma that allows to subtract and add one.

lemma `Int_ZF_4_L1A:`
  assumes `m∈int` **shows** `m $- $#1 $+ $#1 = m`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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)`
  ⟨*proof*⟩

Integer intervals are finite.

lemma **(in** `int0)` `Int_ZF_4_L5:` **assumes** `A1: i∈ℤ k∈ℤ`
  shows `i..k ∈ Fin(ℤ)`
⟨*proof*⟩

Bounded integer sets are finite.

lemma **(in** `int0)` `Int_ZF_4_L6:` **assumes** `A1: IsBounded(A,IntegerOrder)`
  shows `A ∈ Fin(ℤ)`
⟨*proof*⟩

A subset of integers is bounded iff it is finite.

theorem **(in** `int0)` `Int_bounded_iff_fin:`
  shows `IsBounded(A,IntegerOrder)⟷ A∈Fin(ℤ)`
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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*⟩

end

55 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.

55.1 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
  ⟨*proof*⟩

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*⟩

Zero and one are integers.

lemma (**in** int0) int_zero_one_are_int: **shows** $\mathbf{0} \in \mathbb{Z}$ $\mathbf{1} \in \mathbb{Z}$
⟨*proof*⟩

Negative of zero is zero.

lemma (**in** int0) int_zero_one_are_intA: **shows** $(-\mathbf{0}) = \mathbf{0}$
⟨*proof*⟩

Properties with one integer.

lemma (**in** int0) Int_ZF_1_1_L4: **assumes** A1: $a \in \mathbb{Z}$
shows
$a+\mathbf{0} = a$
$\mathbf{0}+a = a$
$a\cdot\mathbf{1} = a$ $\mathbf{1}\cdot a = a$
$\mathbf{0}\cdot a = \mathbf{0}$ $a\cdot\mathbf{0} = \mathbf{0}$
$(-a) \in \mathbb{Z}$ $(-(-a)) = a$
$a-a = \mathbf{0}$ $a-\mathbf{0} = a$ $\mathbf{2}\cdot a = a+a$
⟨*proof*⟩

Properties that require two integers.

lemma (**in** int0) Int_ZF_1_1_L5: **assumes** $a\in\mathbb{Z}$ $b\in\mathbb{Z}$
shows
$a+b \in \mathbb{Z}$
$a-b \in \mathbb{Z}$
$a\cdot b \in \mathbb{Z}$
$a+b = b+a$
$a\cdot b = b\cdot a$
$(-b)-a = (-a)-b$
$(-(a+b)) = (-a)-b$
$(-(a-b)) = ((-a)+b)$
$(-a)\cdot b = -(a\cdot b)$
$a\cdot(-b) = -(a\cdot b)$
$(-a)\cdot(-b) = a\cdot b$
⟨*proof*⟩

2 and 3 are integers.

lemma (**in** int0) int_two_three_are_int: **shows** $\mathbf{2} \in \mathbb{Z}$ $\mathbf{3} \in \mathbb{Z}$
⟨*proof*⟩

Another property with two integers.

lemma (**in** int0) Int_ZF_1_1_L5B:
assumes $a\in\mathbb{Z}$ $b\in\mathbb{Z}$
shows $a-(-b) = a+b$
⟨*proof*⟩

Properties that require three integers.

lemma (**in** int0) Int_ZF_1_1_L6: **assumes** $a\in\mathbb{Z}$ $b\in\mathbb{Z}$ $c\in\mathbb{Z}$
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
```
⟨*proof*⟩

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
⟨*proof*⟩

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)
```
⟨*proof*⟩

55.2 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**)
⟨*proof*⟩

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*⟩

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
```
⟨*proof*⟩

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*⟩

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*⟩

A formula with a nonpositive integer.

lemma **(in** int0) Int_ZF_1_2_L4: **assumes** a≤**0**
  shows abs(a)+**1** = abs(a-**1**)
  ⟨*proof*⟩

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*⟩

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*⟩

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
  ⟨*proof*⟩

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**
  ⟨*proof*⟩

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

55.3 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: $\mathbf{0}\leq$a $\mathbf{0}\leq$b
  and A3: $\mathbf{0}$ $\leq$ a·b
  shows $\mathbf{0}$ $\leq$ a·(b+$\mathbf{1}$)
⟨*proof*⟩

Product of nonnegative integers is nonnegative.

lemma (**in** int0) Int_ZF_1_3_L2: **assumes** A1: $\mathbf{0}\leq$a $\mathbf{0}\leq$b
  shows $\mathbf{0}\leq$a·b
⟨*proof*⟩

The set of nonnegative integers is closed under multiplication.

lemma (**in** int0) Int_ZF_1_3_L2A: **shows**
  $\mathbb{Z}^+$ {is closed under} IntegerMultiplication
⟨*proof*⟩

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($\mathbb{Z}$,IntegerAddition,IntegerMultiplication,IntegerOrder)
  ring1($\mathbb{Z}$,IntegerAddition,IntegerMultiplication,IntegerOrder)
  ⟨*proof*⟩

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: $\mathbf{1}\leq$a $\mathbf{1}\leq$b
  and A2: $\mathbf{1}$ $\leq$ a·b
  shows $\mathbf{1}$ $\leq$ a·(b+$\mathbf{1}$)
⟨*proof*⟩

Product of integers that are greater that one is greater than one.

lemma (**in** int0) Int_ZF_1_3_L3:
  assumes A1: $\mathbf{1}\leq$a $\mathbf{1}\leq$b
  shows $\mathbf{1}$ $\leq$ a·b
⟨*proof*⟩

$|a\cdot(-b)| = |(-a)\cdot b| = |(-a)\cdot(-b)| = |a\cdot b|$ This is a property of ordered rings..

lemma (**in** int0) Int_ZF_1_3_L4: **assumes** a$\in\mathbb{Z}$ b$\in\mathbb{Z}$
  shows
  abs((-a)·b) = abs(a·b)
  abs(a·(-b)) = abs(a·b)
  abs((-a)·(-b)) = abs(a·b)
  ⟨*proof*⟩

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)
  ⟨proof⟩
```

Double nonnegative is nonnegative. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L5A: assumes 0≤a
  shows 0≤2·a
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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
  ⟨*proof*⟩

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**)
  ⟨*proof*⟩

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*⟩

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**
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

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*⟩

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**
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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∈$\mathbb{Z}_+$
shows
a·c ≤ b·c
c·a ≤ c·b
⟨*proof*⟩

A rearrangement with four integers and absolute value.

lemma (in int0) Int_ZF_1_3_L14:
assumes A1: a∈$\mathbb{Z}$ b∈$\mathbb{Z}$ c∈$\mathbb{Z}$ d∈$\mathbb{Z}$
shows abs(a·b)+(abs(a)+c)·d = (d+abs(b))·abs(a)+c·d
⟨*proof*⟩

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∈$\mathbb{Z}$ n∈$\mathbb{Z}$
and A2: ¬(abs(m) ≤ abs(n))
shows n ≤ abs(m) m≠**0**
⟨*proof*⟩

Negative of a nonnegative is nonpositive.

lemma (**in** int0) Int_ZF_1_3_L16: **assumes** A1: **0** $\leq$ m
  shows (-m) $\leq$ **0**
⟨*proof*⟩

Some statements about intervals centered at 0.

lemma (**in** int0) Int_ZF_1_3_L17: **assumes** A1: m∈ℤ
  shows
  (-abs(m)) $\leq$ abs(m)
  (-abs(m))..abs(m) $\neq$ 0
⟨*proof*⟩

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 $\leq$ GreaterOf(IntegerOrder,m,n)
  n $\leq$ GreaterOf(IntegerOrder,m,n)
  SmallerOf(IntegerOrder,m,n) $\leq$ m
  SmallerOf(IntegerOrder,m,n) $\leq$ n
  ⟨*proof*⟩

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) $\leq$ n
  shows
  (-n) $\leq$ m m $\leq$ n
  m ∈ (-n)..n
  0 $\leq$ n
  ⟨*proof*⟩

A slight generalization of the above lemma.

lemma (**in** int0) Int_ZF_1_3_L19A:
  assumes A1: m∈ℤ **and** A2: abs(m) $\leq$ n **and** A3: **0**≤k
  shows (-(n+k)) $\leq$ m
  ⟨*proof*⟩

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)) $\leq$ L
  shows IsBounded({b(x). x∈X},IntegerOrder)
⟨*proof*⟩

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) $\leq$ L
  ⟨*proof*⟩

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

55.4 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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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})
  ⟨proof⟩
```

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⟩
```

55.5 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 $\mathbb{Z}_+$ {is closed under} IntegerAddition
  ⟨*proof*⟩

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∈$\mathbb{Z}_+$ b∈$\mathbb{Z}_+$ **shows** a+b ∈ $\mathbb{Z}_+$
  ⟨*proof*⟩

$\mathbb{Z}^+$ is bounded below.

lemma (**in** int0) Int_ZF_1_5_L1: **shows**
  IsBoundedBelow($\mathbb{Z}^+$,IntegerOrder)
  IsBoundedBelow($\mathbb{Z}_+$,IntegerOrder)
  ⟨*proof*⟩

Subsets of $\mathbb{Z}^+$ are bounded below.

lemma (**in** int0) Int_ZF_1_5_L1A: **assumes** A ⊆ $\mathbb{Z}^+$
  shows IsBoundedBelow(A,IntegerOrder)
  ⟨*proof*⟩

Subsets of $\mathbb{Z}_+$ are bounded below.

lemma (**in** int0) Int_ZF_1_5_L1B: **assumes** A1: A ⊆ $\mathbb{Z}_+$
  shows IsBoundedBelow(A,IntegerOrder)
  ⟨*proof*⟩

Every nonempty subset of positive integers has a mimimum.

lemma (**in** int0) Int_ZF_1_5_L1C: **assumes** A ⊆ $\mathbb{Z}_+$ **and** A ≠ 0
  shows
  HasAminimum(IntegerOrder,A)
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  ⟨*proof*⟩

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 ⊆ $\mathbb{Z}^+$ **and** A2: A ∉ Fin($\mathbb{Z}$) **and** A3: D∈$\mathbb{Z}$
  shows ∃n∈A. D≤n
⟨*proof*⟩

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 ⊆ $\mathbb{Z}_+$ **and** A2: A ∉ Fin($\mathbb{Z}$) **and** A3: D∈$\mathbb{Z}$
  shows ∃n∈A. D≤n
⟨*proof*⟩

An integer is either positive, zero, or its opposite is postitive.

lemma (**in** int0) Int_decomp: **assumes** m∈$\mathbb{Z}$
  shows Exactly_1_of_3_holds (m=**0**,m∈$\mathbb{Z}_+$,(-m)∈$\mathbb{Z}_+$)
  ⟨*proof*⟩

An integer is zero, positive, or it's inverse is positive.

lemma (**in** int0) int_decomp_cases: **assumes** m∈$\mathbb{Z}$
  shows m=**0** ∨ m∈$\mathbb{Z}_+$ ∨ (-m) ∈ $\mathbb{Z}_+$
  ⟨*proof*⟩

An integer is in the positive set iff it is greater or equal one.

lemma (**in** int0) Int_ZF_1_5_L3: **shows** m∈$\mathbb{Z}_+$ ⟷ **1**≤m
⟨*proof*⟩

The set of positive integers is closed under multiplication. The unfolded form.

lemma (**in** int0) pos_int_closed_mul_unfold:
  assumes a∈$\mathbb{Z}_+$ b∈$\mathbb{Z}_+$
  shows a·b ∈ $\mathbb{Z}_+$
  ⟨*proof*⟩

The set of positive integers is closed under multiplication.

lemma (**in** int0) pos_int_closed_mul: **shows**
  $\mathbb{Z}_+$ {is closed under} IntegerMultiplication
  ⟨*proof*⟩

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($\mathbb{Z}$,IntegerAddition,IntegerMultiplication)
  ⟨*proof*⟩

Nonnegative integers are positive ones plus zero.

lemma (**in** int0) Int_ZF_1_5_L3A: **shows** $\mathbb{Z}^+$ = $\mathbb{Z}_+$ ∪ {**0**}
  ⟨*proof*⟩

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: $f:\mathbb{Z}\rightarrow\mathbb{Z}$ **and** A2: $K\in\mathbb{Z}$ $N\in\mathbb{Z}$
  shows $\exists C\in\mathbb{Z}.\ \forall n\in\mathbb{Z}_+.\ K \leq f(n) + C \longrightarrow N\leq n$
$\langle proof \rangle$

Absolute value is identity on positive integers.

lemma (in int0) Int_ZF_1_5_L4A:
  assumes $a\in\mathbb{Z}_+$ **shows** abs(a) = a
  $\langle proof \rangle$

One and two are in $\mathbb{Z}_+$.

lemma (in int0) int_one_two_are_pos: **shows** $\mathbf{1} \in \mathbb{Z}_+$ $\mathbf{2} \in \mathbb{Z}_+$
  $\langle proof \rangle$

The image of $\mathbb{Z}_+$ by a function defined on integers is not empty.

lemma (in int0) Int_ZF_1_5_L5: **assumes** A1: $f : \mathbb{Z}\rightarrow X$
  shows $f(\mathbb{Z}_+) \neq 0$
$\langle proof \rangle$

If n is positive, then $n - 1$ is nonnegative.

lemma (in int0) Int_ZF_1_5_L6: **assumes** A1: $n \in \mathbb{Z}_+$
  shows
  $\mathbf{0} \leq n-\mathbf{1}$
  $\mathbf{0} \in \mathbf{0}..(n-\mathbf{1})$
  $\mathbf{0}..(n-\mathbf{1}) \subseteq \mathbb{Z}$
$\langle proof \rangle$

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** $a \in \mathbb{Z}_+$ **and** $a\leq b$
  shows $b \in \mathbb{Z}_+$
$\langle proof \rangle$

Adding a positive integer increases integers.

lemma (in int0) Int_ZF_1_5_L7A: **assumes** $a\in\mathbb{Z}$ $b \in \mathbb{Z}_+$
  shows $a \leq a+b$ $a \neq a+b$ $a+b \in \mathbb{Z}$
  $\langle proof \rangle$

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** $a\in\mathbb{Z}$
  shows
  $a \leq$ GreaterOf(IntegerOrder,$\mathbf{1}$,a)
  GreaterOf(IntegerOrder,$\mathbf{1}$,a) $\in \mathbb{Z}_+$
  GreaterOf(IntegerOrder,$\mathbf{1}$,a) + $\mathbf{1} \in \mathbb{Z}_+$

```
  a ≤ GreaterOf(IntegerOrder,1,a) + 1
  a ≠ GreaterOf(IntegerOrder,1,a) + 1
  ⟨proof⟩
```

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₊
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

55.6 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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

55.7 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*⟩

A lemma about splitting (not really, there is some overlap) the ℤ×ℤ into six subsets (cases). The subsets are as follows: first and third qaudrant, 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**
  ⟨*proof*⟩

end

56 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.

56.1 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)
  ⟨proof⟩
```

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⟩
```

$(m \cdot k)$ 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
  ⟨proof⟩
```

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⟩
```

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⟩
```

end

57 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.

57.1 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 (𝒮 )
  defines slopes_def[simp]: 𝒮 ≡ AlmostHoms(ℤ,IntegerAddition)

  fixes posslopes (𝒮₊)
  defines posslopes_def[simp]: 𝒮₊ ≡ {s∈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]: m^S ≡ {⟨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)
  ⟨*proof*⟩

Type information related to the homomorphism difference expression.

lemma (in int1) Int_ZF_2_1_L2: **assumes** f∈$\mathcal{S}$ **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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

Slopes map integers into integers.

lemma (in int1) Int_ZF_2_1_L2B:
  assumes A1: f∈$\mathcal{S}$ **and** A2: m∈ℤ
  shows f(m) ∈ ℤ
⟨*proof*⟩

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)
  ⟨*proof*⟩

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∈$\mathcal{S}$ **and** A2: m∈ℤ n∈ℤ
  shows
  f(m+n) = f(m)+(f(n)+δ(f,m,n))
⟨*proof*⟩

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
⟨*proof*⟩

Composition of slopes is a slope.

lemma (in int1**)** Int_ZF_2_1_L11:
assumes s∈𝒮 r∈𝒮
shows s∘r ∈ 𝒮
⟨*proof*⟩

Negative of a slope is a slope.

lemma (in int1**)** Int_ZF_2_1_L12: **assumes** s∈𝒮 **shows** -s ∈ 𝒮
⟨*proof*⟩

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))
⟨*proof*⟩

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)
⟨*proof*⟩

Sum of slopes is a slope.

lemma (in int1**)** Int_ZF_2_1_L12C: **assumes** s∈𝒮 r∈𝒮
shows s+r ∈ 𝒮
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

Identity is a slope.

```
lemma (in int1) Int_ZF_2_1_L17: shows id(ℤ) ∈ 𝒮
  ⟨proof⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

57.2 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*⟩

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1) \cdot \mathtt{max}\delta\mathtt{(f)}$. 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⟩
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1) \cdot \mathtt{max}\delta$. 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩

end
```

58 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`.

58.1 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: $f \in \mathcal{S}_+$ **shows** $f:\mathbb{Z}\rightarrow\mathbb{Z}$
⟨*proof*⟩

A small technical lemma to simplify the proof of the next theorem.

lemma (in int1**)** Int_ZF_2_3_L1A:
assumes A1: $f \in \mathcal{S}_+$ **and** A2: $\exists n \in f(\mathbb{Z}_+) \cap \mathbb{Z}_+.\ a \leq n$
shows $\exists M \in \mathbb{Z}_+.\ a \leq f(M)$
⟨*proof*⟩

The next lemma is Lemma 3 in the Arthan's paper.

lemma (in int1**)** Arthan_Lem_3:
assumes A1: $f \in \mathcal{S}_+$ **and** A2: $D \in \mathbb{Z}_+$
shows $\exists M \in \mathbb{Z}_+.\ \forall m \in \mathbb{Z}_+.\ (m+\mathbf{1})\cdot D \leq f(m\cdot M)$
⟨*proof*⟩

A special case of `Arthan_Lem_3` when $D = 1$.

corollary (in int1**)** Arthan_L_3_spec: **assumes** A1: $f \in \mathcal{S}_+$
shows $\exists M \in \mathbb{Z}_+.\forall n \in \mathbb{Z}_+.\ n+\mathbf{1} \leq f(n\cdot M)$
⟨*proof*⟩

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∈$\mathcal{S}$ f ∉ $\mathcal{S}_+$
⟨*proof*⟩

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∈$\mathcal{S}$ **and** A2: f ∉ $\mathcal{S}_+$
shows IsBoundedAbove(f(ℤ$_+$), IntegerOrder)
⟨*proof*⟩

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∈$\mathcal{S}$ **and** A2: -f ∉ $\mathcal{S}_+$
shows IsBoundedBelow(f(ℤ$_+$), IntegerOrder)
⟨*proof*⟩

A slope that is bounded on $\mathbb{Z}_+$ is bounded everywhere.

lemma (in int1**)** Int_ZF_2_3_L4:
assumes A1: f∈$\mathcal{S}$ **and** A2: m∈ℤ
and A3: ∀n∈ℤ$_+$. abs(f(n)) ≤ L
shows abs(f(m)) ≤ **2**·maxδ(f) + L
⟨*proof*⟩

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∈$\mathcal{S}$ **and** A2: IsBounded(f(ℤ$_+$), IntegerOrder)
shows f ∈ FinRangeFunctions(ℤ,ℤ)
⟨*proof*⟩

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∈$\mathcal{S}$ **and** IsBoundedBelow(f(ℤ$_+$), IntegerOrder)
shows f ∈ FinRangeFunctions(ℤ,ℤ) ∨ f∈$\mathcal{S}_+$
⟨*proof*⟩

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∈$\mathcal{S}$ g∈$\mathcal{S}$ **and**
A2: ∀n∈ℤ$_+$. f(n) ≤ g(n)
shows f∼g ∨ g + (-f) ∈ $\mathcal{S}_+$
⟨*proof*⟩

Positive slopes are arbitrarily large for large enough arguments.

lemma (in int1**)** Int_ZF_2_3_L5:
assumes A1: f∈$\mathcal{S}_+$ **and** A2: K∈ℤ

shows $\exists N\in\mathbb{Z}_+ .\ \forall m.\ N\leq m \longrightarrow K \leq f(m)$
⟨*proof*⟩

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: $f\in\mathcal{S}_+$ **and** A2: $K\in\mathbb{Z}$
shows $\exists N\in\mathbb{Z}_+ .\ \forall m.\ N\leq m \longrightarrow f(-m) \leq K$
⟨*proof*⟩

A special case of `Int_ZF_2_3_L5` where $K = 1$.

corollary (**in** int1) Int_ZF_2_3_L6: **assumes** $f\in\mathcal{S}_+$
shows $\exists N\in\mathbb{Z}_+ .\ \forall m.\ N\leq m \longrightarrow f(m) \in \mathbb{Z}_+$
⟨*proof*⟩

A special case of `Int_ZF_2_3_L5` where $m = N$.

corollary (**in** int1) Int_ZF_2_3_L6A: **assumes** $f\in\mathcal{S}_+$ **and** $K\in\mathbb{Z}$
shows $\exists N\in\mathbb{Z}_+ .\ K \leq f(N)$
⟨*proof*⟩

If values of a slope are not bounded above, then the slope is positive.

lemma (**in** int1) Int_ZF_2_3_L7: **assumes** A1: $f\in\mathcal{S}$
and A2: $\forall K\in\mathbb{Z}.\ \exists n\in\mathbb{Z}_+ .\ K \leq f(n)$
shows $f \in \mathcal{S}_+$
⟨*proof*⟩

For unbounded slope f either $f \in\mathcal{S}_+$ of $-f \in\mathcal{S}_+$.

theorem (**in** int1) Int_ZF_2_3_L8:
assumes A1: $f\in\mathcal{S}$ **and** A2: $f \notin$ FinRangeFunctions($\mathbb{Z}$,$\mathbb{Z}$)
shows ($f \in \mathcal{S}_+$) Xor ($(-f) \in \mathcal{S}_+$)
⟨*proof*⟩

The sum of positive slopes is a positive slope.

theorem (**in** int1) sum_of_pos_sls_is_pos_sl:
assumes A1: $f \in \mathcal{S}_+$ $g \in \mathcal{S}_+$
shows $f+g \in \mathcal{S}_+$
⟨*proof*⟩

The composition of positive slopes is a positive slope.

theorem (**in** int1) comp_of_pos_sls_is_pos_sl:
assumes A1: $f \in \mathcal{S}_+$ $g \in \mathcal{S}_+$
shows $f\circ g \in \mathcal{S}_+$
⟨*proof*⟩

A slope equivalent to a positive one is positive.

lemma (**in** int1) Int_ZF_2_3_L9:
assumes A1: $f \in \mathcal{S}_+$ **and** A2: $\langle f,g\rangle \in$ AlEqRel **shows** $g \in \mathcal{S}_+$
⟨*proof*⟩

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,$\mathcal{S}_+$)
⟨*proof*⟩

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∈$\mathcal{S}$ g∈$\mathcal{S}$
  and A2: R = {AlEqRel{s}. s∈$\mathcal{S}_+$}
  and A3: (f∈$\mathcal{S}_+$) Xor (g∈$\mathcal{S}_+$)
  shows (AlEqRel{f} ∈ R) Xor (AlEqRel{g} ∈ R)
⟨*proof*⟩

Identity function is a positive slope.

lemma **(in** int1) Int_ZF_2_3_L11: **shows** id(ℤ) ∈ $\mathcal{S}_+$
⟨*proof*⟩

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*⟩

58.2 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 ∈ $\mathcal{S}_+$ **and** A2: p∈$\mathbb{Z}_+$ **and** A3: A = {n∈$\mathbb{Z}_+$. p ≤ f(n)}
  shows
  A ⊆ $\mathbb{Z}_+$
  A ≠ 0
  $\mathtt{f}^{-1}$(p) ∈ A
  ∀m∈A. $\mathtt{f}^{-1}$(p) ≤ m
⟨*proof*⟩

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))
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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: $\forall m\in\mathbb{Z}_+.\ f^{-1}(m)-\mathbf{1} \in \mathbb{Z}_+$
shows $\exists M.\ \forall x\in\mathbb{Z}_+\times\mathbb{Z}_+.\ abs(\varepsilon(f,x)) \leq M$
⟨*proof*⟩

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: $f \in \mathcal{S}_+$ **and** A2: $g = \{\langle p,f^{-1}(p)\rangle.\ p\in\mathbb{Z}_+\}$
shows
$g : \mathbb{Z}_+\rightarrow\mathbb{Z}_+$
$g : \mathbb{Z}_+\rightarrow\mathbb{Z}$
⟨*proof*⟩

What are the values of the (candidate for) the inverse of a positive slope?

lemma (in int1**)** Int_ZF_2_4_L10:
assumes A1: $f \in \mathcal{S}_+$ **and** A2: $g = \{\langle p,f^{-1}(p)\rangle.\ p\in\mathbb{Z}_+\}$ **and** A3: $p\in\mathbb{Z}_+$
shows $g(p) = f^{-1}(p)$
⟨*proof*⟩

The (candidate for) the inverse of a positive slope is a slope.

lemma (in int1**)** Int_ZF_2_4_L11: **assumes** A1: $f \in \mathcal{S}_+$ **and**
A2: $\forall m\in\mathbb{Z}_+.\ f^{-1}(m)-\mathbf{1} \in \mathbb{Z}_+$ **and**
A3: $g = \{\langle p,f^{-1}(p)\rangle.\ p\in\mathbb{Z}_+\}$
shows OddExtension($\mathbb{Z}$,IntegerAddition,IntegerOrder,g) $\in \mathcal{S}$
⟨*proof*⟩

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: $f \in \mathcal{S}_+$ **and**
A2: $\forall m\in\mathbb{Z}_+.\ f^{-1}(m)-\mathbf{1} \in \mathbb{Z}_+$
shows $\exists h\in\mathcal{S}.\ f\circ h \sim id(\mathbb{Z})$
⟨*proof*⟩

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: $f \in \mathcal{S}_+$
shows $\exists g\in\mathcal{S}.\ f\sim g \wedge (\exists h\in\mathcal{S}.\ g\circ h \sim id(\mathbb{Z}))$
⟨*proof*⟩

58.3 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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: (m^S) ∼ (k^S)
  shows m=k
⟨proof⟩
```

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
  1^S = id(ℤ)
  0^S ∈ FinRangeFunctions(ℤ,ℤ)
⟨proof⟩
```

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⟩
```

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⟩

end


theory IntModule_ZF imports Module_ZF Int_ZF_1 Group_ZF

begin
```

59 $\mathbb{Z}$ 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 action_unique:
  assumes IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,f,S1)
and IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,f,S2)
  shows S1 = S2
⟨*proof*⟩

The action we will show works is $n \mapsto (g \mapsto g^n)$. It is a well-defined function

lemma(in abelian_group) group_action_int_fun:
  defines S ≡ {⟨$# n,{⟨x,Fold(P,**1**,n×{x})⟩. x∈G}⟩. n∈nat}∪ {⟨$- $# n,GroupInv(G,P) O {⟨x,Fold(P,**1**, n×{x})⟩. x∈G}⟩. n∈nat}
  shows S:int → End(G,P) ⟨*proof*⟩

The action is defined on positive and negative numbers by the following folds:

lemma(in abelian_group) group_action_int_dest:
  defines S ≡ {⟨$# n,{⟨x,Fold(P,**1**,n×{x})⟩. x∈G}⟩. n∈nat}∪ {⟨$- $# n,GroupInv(G,P) O {⟨x,Fold(P,**1**, n×{x})⟩. x∈G}⟩. n∈nat}
  assumes n∈nat x∈G
  shows (S($#n))x =Fold(P,**1**,n×{x}) (S($- $#n))x =Fold(P,**1**,n×{x})$^{-1}$
⟨*proof*⟩

The action takes 1 to the identity endomorphism

lemma(in abelian_group) group_action_int_unit:
  defines S ≡ {⟨$# n,{⟨x,Fold(P,**1**,n×{x})⟩. x∈G}⟩. n∈nat}∪ {⟨$- $# n,GroupInv(G,P) O {⟨x,Fold(P,**1**, n×{x})⟩. x∈G}⟩. n∈nat}
  shows STheNeutralElement(int,IntegerMultiplication) = TheNeutralElement(End(G, P), Composition(G) {in End} [G,P])
⟨*proof*⟩

59.1 Fold formulas

Folding the sum of 2 numbers is equivalent to doing 2 folds

lemma(in abelian_group) group_action_int_add:
  assumes z1∈nat z2∈nat g∈G
  shows Fold(P,**1**,(z1#+z2)×{g}) = Fold(P,**1**,(z1)×{g})·Fold(P,**1**,(z2)×{g})
⟨*proof*⟩

The element on the fold, can commute with the `Fold`

corollary(in abelian_group) group_action_int_comm:
  assumes z1∈nat g∈G
  shows g·Fold(P,**1**,z1×{g}) = Fold(P,**1**,z1×{g})·g
⟨*proof*⟩

Folding an inversed element is equivalent of folding and the inverting

```
lemma(in abelian_group) group_action_int_inv:
  assumes z∈nat g∈G
  shows Fold(P,1,z×{g⁻¹}) = Fold(P,1,z×{g})⁻¹
⟨proof⟩
```

Folds when considering a well defined substraction

```
lemma(in abelian_group) group_action_int_minus:
  assumes z1∈nat z2∈nat g∈G z2 ≤ z1
  shows Fold(P,1,(z1#-z2)×{g}) = Fold(P,1,(z1)×{g})·Fold(P,1,(z2)×{g})⁻¹
⟨proof⟩
```

Fold negative number by substraction

```
lemma(in abelian_group) group_action_int_minus_rev:
  assumes z1∈nat z2∈nat g∈G z1 ≤ z2
  shows Fold(P,1,(z2 #- z1)×{g})⁻¹ = Fold(P,1,z1×{g})·Fold(P,1,z2×{g})⁻¹
⟨proof⟩
```

The action is an group homomorphism between $(\mathbb{Z}, +)$ and (G, P)

```
lemma(in abelian_group) group_action_int_add_morphism:
  defines S ≡ {⟨$# n,{⟨x,Fold(P,1,n×{x})⟩. x∈G}⟩. n∈nat}∪ {⟨$- $# n,GroupInv(G,P)
O {⟨x,Fold(P,1, n×{x})⟩. x∈G}⟩. n∈nat}
  shows ∀r∈int. ∀s∈int. ∀g∈G. S  (IntegerAddition ⟨r, s⟩)  g = P  ⟨(S
r)  g, (S  s)  g⟩
⟨proof⟩
```

Same as before, but not pointwise

```
lemma(in abelian_group) group_action_int_add_morphism_fun:
  defines S ≡ {⟨$# n,{⟨x,Fold(P,1,n×{x})⟩. x∈G}⟩. n∈nat} ∪ {⟨$- $# n,GroupInv(G,P)
O {⟨x,Fold(P,1, n×{x})⟩. x∈G}⟩. n∈nat}
  shows ∀r∈int. ∀s∈int. S  (IntegerAddition ⟨r, s⟩) = EndAdd(G,P)  ⟨(S
r), (S  s)⟩
⟨proof⟩
```

Fold of a multiplication

```
lemma(in abelian_group) group_action_int_mult:
  assumes z1∈nat z2∈nat g∈G
  shows Fold(P,1,(z1#*z2)×{g}) = Fold(P,1,z2×{Fold(P,1,z1×{g})})
⟨proof⟩
```

Multiplying 2 int_of natural numbers, is the same as multiplying the natural numbers and then applying int_of

```
lemma int_of_mult:
  assumes nr:nat ns:nat
  shows ($# nr) $* ($# ns) = $# (nr #* ns)
⟨proof⟩
```

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 ≡ {⟨$# n,{⟨x,Fold(P,1,n×{x})⟩. x∈G}⟩. n∈nat} ∪ {⟨$- $# n,GroupInv(G,P)
O {⟨x,Fold(P,1, n×{x})⟩. x∈G}⟩. n∈nat}
  shows ∀r∈int. ∀s∈int. S  (IntegerMultiplication ⟨r, s⟩) = EndMult(G,P)⟨Sr,Ss⟩
⟨proof⟩
```

The action defines a module

```
theorem(in abelian_group) group_action_int:
  defines S ≡ {⟨$# n,{⟨x,Fold(P,1,n×{x})⟩. x∈G}⟩. n∈nat} ∪ {⟨$- $# n,GroupInv(G,P)
O {⟨x,Fold(P,1, n×{x})⟩. x∈G}⟩. n∈nat}
  shows IsLeftModule(int,IntegerAddition,IntegerMultiplication,G,P,S)
⟨proof⟩
```

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={⟨$# n,{⟨x,Fold(P,1,n×{x})⟩. x∈G}⟩. n∈nat} ∪ {⟨$- $# n,GroupInv(G,P)
O {⟨x,Fold(P,1, n×{x})⟩. x∈G}⟩. n∈nat}
  ⟨proof⟩
```

New assumption to consider integers and an abelian group

```
locale abelian_group_int_action = abelian_group + int0
```

Under this assumptions, we have an action

```
sublocale abelian_group_int_action < int_action:module0 ints IntegerAddition
IntegerMultiplication
  ia iminus isub imult izero ione itwo λq. imult(q,q) G P {⟨$# n,{⟨x,Fold(P,neut,n×{x})⟩.
x∈G}⟩. n∈nat} ∪ {⟨$- $# n,GroupInv(G,P) O {⟨x,Fold(P,neut, n×{x})⟩. x∈G}⟩.
n∈nat}
  neut groper λs g. ({⟨$# n,{⟨x,Fold(P,neut,n×{x})⟩. x∈G}⟩. n∈nat} ∪ {⟨$-
$# n,GroupInv(G,P) O {⟨x,Fold(P,neut, n×{x})⟩. x∈G}⟩. n∈nat})sg inv
  λg h. groper(g,inv(h))
  ⟨proof⟩
```

```
abbreviation (in abelian_group_int_action) zone (1ℤ) where
1ℤ ≡ ione
```

Then, the unit in the abelian group

```
abbreviation (in abelian_group_int_action) gone (1G) where
1G ≡ neut
```

```
end
```

60 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.

60.1 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` $\equiv$ `AlmostHoms(int,IntegerAddition)`

The first operation on slopes (pointwise addition) is a special case of the first operation on almost homomorphisms.

definition
  `SlopeOp1` $\equiv$ `AlHomOp1(int,IntegerAddition)`

The second operation on slopes (composition) is a special case of the second operation on almost homomorphisms.

definition
  `SlopeOp2` $\equiv$ `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)`
⟨*proof*⟩

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*⟩

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*⟩

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)`
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨proof⟩
```

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)
⟨proof⟩
```

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`
⟨*proof*⟩

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
⟨proof⟩
```

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`
⟨*proof*⟩

`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)
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

end

61 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.

61.1 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 ≡ {SlopeEquivalenceRel{s}. s ∈ 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 ≡ 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 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 ∼ 68)
defines AlEq_def[simp]: s ∼ r ≡ ⟨s,r⟩ ∈ SlopeEquivalenceRel

fixes slope_add (infix + 70)
defines slope_add_def[simp]:
s + r ≡  SlopeOp1⟨s,r⟩

fixes slope_comp (infix ∘ 71)
defines slope_comp_def[simp]: s ∘ r ≡  SlopeOp2⟨s,r⟩

fixes slopes (𝒮)
defines slopes_def[simp]: 𝒮 ≡ AlmostHoms(int,IntegerAddition)

fixes posslopes (𝒮₊)
defines posslopes_def[simp]: 𝒮₊ ≡ 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}`·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·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 (1_Z)
defines ione_def[simp]: 1_Z ≡ TheNeutralElement(int,IntegerMultiplication)

fixes itwo (2_Z)
defines itwo_def[simp]: 2_Z ≡ 1_Z+1_Z

fixes abs
defines abs_def[simp]:
abs(m) ≡ AbsoluteValue(int,IntegerAddition,IntegerOrder)(m)

fixes δ
defines δ_def[simp]: δ(s,m,n) ≡ s(m+n)-s(m)-s(n)
```

61.2 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⟩
```

Classes of slopes are real numbers.

```
lemma (in real1) Real_ZF_1_1_L3: assumes A1: f ∈ S
  shows [f] ∈ ℝ
⟨proof⟩
```

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⟩
```

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⟩
```

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]
  ⟨proof⟩
```

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⟩
```

Multiplication of real numbers is commutative.

```
theorem (in real1) real_mult_commute: assumes A1: a∈ℝ  b∈ℝ
  shows a·b = b·a
⟨proof⟩
```

Multiplication is commutative on reals.

```
lemma real_mult_commutative: shows
  RealMultiplication {is commutative on} RealNumbers
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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]
  ⟨*proof*⟩

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) ∈ $\mathcal{S}$
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

Zero is not one.

lemma (**in** real1) real_zero_not_one: **shows** **1**≠**0**
⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

61.3 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*⟩

Positive reals are the same as classes of a positive slopes.

lemma (**in** real1) Real_ZF_1_2_L2:
  shows a ∈ PositiveReals ⟷ (∃f∈$\mathcal{S}_+$. a = [f])
⟨*proof*⟩

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∈$\mathcal{S}_+$ g∈$\mathcal{S}_+$
  shows
  f+g ∈ $\mathcal{S}_+$
  f∘g ∈ $\mathcal{S}_+$
  ⟨*proof*⟩

Bounded integer maps are not positive slopes.

lemma (**in** real1) Real_ZF_1_2_L5:
  assumes f ∈ BoundedIntMaps
  shows f ∉ $\mathcal{S}_+$
  ⟨*proof*⟩

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*⟩

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) ∈ $\mathcal{S}_+$)
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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
  ⟨*proof*⟩

A sufficient condition for two classes to be in the real order.

lemma **(in** real1) Real_ZF_1_2_L12: **assumes** A1: f ∈ $\mathcal{S}$ g ∈ $\mathcal{S}$ **and**
  A2: f∼g ∨ (g + (-f)) ∈ $\mathcal{S}_+$
  shows [f] ≤ [g]
⟨*proof*⟩

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
  ⟨*proof*⟩

Real order is antisymmetric.

lemma **(in** real1) real_ord_antisym:
  assumes A1: a≤b b≤a **shows** a=b
⟨*proof*⟩

Real order is transitive.

lemma **(in** real1) real_ord_transitive: **assumes** A1: a≤b b≤c
  shows a≤c
⟨*proof*⟩

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
  ⟨*proof*⟩

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∈$\mathbb{R}_+$
  shows c·a ≤ c·b
  ⟨*proof*⟩

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
  ⟨*proof*⟩

$a \leq b$ implies that $0 \leq b - a$.

lemma (**in** real1) Real_ZF_1_2_L16: **assumes** a≤b
  shows **0** ≤ b-a
  ⟨*proof*⟩

A sum of nonnegative elements is nonnegative.

lemma (**in** real1) Real_ZF_1_2_L17: **assumes** **0**≤a **0**≤b
  shows **0** ≤ a+b
  ⟨*proof*⟩

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
  ⟨*proof*⟩

The order on real is reflexive.

lemma (**in** real1) real_ord_refl: **assumes** a∈ℝ **shows** a≤a
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

The order on reals is a relation on reals.

lemma **(in** real1) Real_ZF_1_2_L22: **shows** OrderOnReals ⊆ ℝ×ℝ
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

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*⟩

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 ∈ $\mathbb{R}_+$ **and** a≤b **and** 1<c
  shows a<b·c
  ⟨*proof*⟩

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
  ⟨proof⟩
```

Real order is translation invariant.

```
lemma (in real1) real_ord_transl_inv:
  assumes a≤b and c∈ℝ
  shows c+a ≤ c+b
  ⟨proof⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

61.4 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** $f \in \mathcal{S}_+$
  shows $\exists g\in\mathcal{S}.\ f\sim g \wedge (\exists h\in\mathcal{S}.\ g\circ h \sim id(int))$
  ⟨*proof*⟩

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*⟩

Reals form a field.

lemma reals_are_field:
  shows IsAfield(RealNumbers,RealAddition,RealMultiplication)
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

If a is positive, then a^{-1} is also positive.

lemma (**in** real1) Real_ZF_1_3_L1: **assumes** $a \in \mathbb{R}_+$
  shows $a^{-1} \in \mathbb{R}_+$ $a^{-1} \in \mathbb{R}$
  ⟨*proof*⟩

A technical fact about multiplying strict inequality by the inverse of one of the sides.

lemma (**in** real1) Real_ZF_1_3_L2:
  assumes $a \in \mathbb{R}_+$ **and** $a^{-1} < b$
  shows $\mathbf{1} < b\cdot a$
  ⟨*proof*⟩

If a is smaller than b, then $(b-a)^{-1}$ is positive.

lemma (**in** real1) Real_ZF_1_3_L3: **assumes** $a<b$
  shows $(b-a)^{-1} \in \mathbb{R}_+$
  ⟨*proof*⟩

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}
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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}
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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}
  ⟨*proof*⟩

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}
  ⟨*proof*⟩

An identity with three real numbers, inverse and cancelling.

lemma (**in** real1) Real_ZF_1_3_L8: **assumes**$a\in\mathbb{R}$ $b\in\mathbb{R}$ $b\neq\mathbf{0}$ $c\in\mathbb{R}$
  shows $a\cdot b\cdot(c\cdot b^{-1}) = a\cdot c$
  ⟨*proof*⟩

61.5 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** $m \in$ int
  shows $m^R \in \mathbb{R}$
  ⟨*proof*⟩

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** $m \in$ int
  shows $(-m)^R = -(m^R)$
  ⟨*proof*⟩

The embedding of sum of integers is the sum of embeddings.

lemma (**in** real1) Real_ZF_1_4_L1A: **assumes** $m \in$ int $k \in$ int
  shows $m^R + k^R = ((m+k)^R)$
  ⟨*proof*⟩

The embedding of a difference of integers is the difference of embeddings.

lemma (**in** real1) Real_ZF_1_4_L1B: **assumes** A1: $m \in$ int $k \in$ int
  shows $m^R - k^R = (m-k)^R$
⟨*proof*⟩

The embedding of the product of integers is the product of embeddings.

lemma (**in** real1) Real_ZF_1_4_L1C: **assumes** $m \in$ int $k \in$ int
  shows $m^R \cdot k^R = (m\cdot k)^R$
  ⟨*proof*⟩

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: $a\in\mathbb{R}$
  shows $\exists m\in$int. $a \leq m^R$
⟨*proof*⟩

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*⟩

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*⟩

The embedding of integers preserves the order.

lemma (**in** real1) Real_ZF_1_4_L5: **assumes** A1: m≤k
  shows m^R ≤ k^R
⟨*proof*⟩

The embedding of integers preserves the strict order.

lemma (**in** real1) Real_ZF_1_4_L5A: **assumes** A1: m≤k m≠k
  shows m^R < k^R
⟨*proof*⟩

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∈$\mathbb{Z}_+$. a < n^R
⟨*proof*⟩

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*⟩

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*⟩

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⟩

Integer zero and one embed as real zero and one.

lemma (in real1) int_0_1_are_real_zero_one:
  shows $\mathbf{0}_Z{}^R$ = **0** $\mathbf{1}_Z{}^R$ = **1**
  ⟨proof⟩

Integer two embeds as the real two.

lemma (in real1) int_two_is_real_two: **shows** $\mathbf{2}_Z{}^R$ = **2**
⟨proof⟩

A positive integer embeds as a positive (hence nonnegative) real.

lemma (in real1) int_pos_is_real_pos: **assumes** A1: p∈$\mathbb{Z}_+$
  shows
  p^R ∈ ℝ
  0 ≤ p^R
  p^R ∈ $\mathbb{R}_+$
⟨proof⟩

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 ∈ $\mathbb{R}_+$
  shows ∃M∈$\mathbb{Z}_+$. x < M^R·y
⟨proof⟩

Taking the floor function preserves the order.

lemma (in real1) Real_ZF_1_4_L9: **assumes** A1: a≤b
  shows ⌊a⌋ ≤ ⌊b⌋
⟨proof⟩

If S is bounded above and p is a positive intereger, 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∈$\mathbb{Z}_+$
  shows
  IsBoundedAbove({⌊p^R·x⌋. x∈S},IntegerOrder)
  Γ(S,p) ∈ {⌊p^R·x⌋. x∈S}
  Γ(S,p) ∈ int
⟨proof⟩

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∈$\mathbb{Z}_+$
  shows ⌊p^R·x⌋ ≤ Γ(S,p)
⟨proof⟩

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⟩
```

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⟩
```

Floor of (real) zero is (integer) zero.

```
lemma (in real1) floor_01_is_zero_one: shows
  ⌊0⌋ = 0_Z    ⌊1⌋ = 1_Z
⟨proof⟩
```

Floor of (real) two is (integer) two.

```
lemma (in real1) floor_2_is_two: shows ⌊2⌋ = 2_Z
⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

A lemma about adding one to floor.

lemma (**in** real1) Real_ZF_1_4_L17A: **assumes** A1: $a\in\mathbb{R}$
shows $\mathbf{1}$ + $\lfloor a \rfloor^R$ = $(\mathbf{1}_Z + \lfloor a \rfloor)^R$
⟨*proof*⟩

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: $a\in\mathbb{R}$
shows
$a - \lfloor a \rfloor^R < \mathbf{1}$
$a < (\mathbf{1}_Z + \lfloor a \rfloor)^R$
⟨*proof*⟩

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 $\exists M\in int.\ \exists N\in\mathbb{Z}_+.\ \ x\cdot N^R < M^R \wedge M^R < y\cdot N^R$
⟨*proof*⟩

Some estimates for the homomorphism difference of the floor function.

lemma (**in** real1) Real_ZF_1_4_L18: **assumes** A1: $x\in\mathbb{R}$ $y\in\mathbb{R}$
shows
$abs(\lfloor x+y \rfloor - \lfloor x \rfloor - \lfloor y \rfloor) \leq \mathbf{2}_Z$
⟨*proof*⟩

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) $S\neq 0$ **and**
A2: $n\in\mathbb{Z}_+$ $x\in S$ **and**
A3: $\Gamma(S,n) = \lfloor n^R\cdot x \rfloor$ **and**
A4: $y\in S$ $x\leq y$
shows $\Gamma(S,n) = \lfloor n^R\cdot y \rfloor$
⟨*proof*⟩

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) $S\neq 0$ **and**
A2: $m\in\mathbb{Z}_+$ $n\in\mathbb{Z}_+$
shows $abs(\Gamma(S,m+n) - \Gamma(S,m) - \Gamma(S,n)) \leq \mathbf{2}_Z$
⟨*proof*⟩

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⌋
  ⟨proof⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩

end
```

62 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.

62.1 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*⟩

Empty set is open.

lemma `empty_open`:
  assumes `T {is a topology}` **shows** `0 ∈ T`
⟨*proof*⟩

The carrier is open.

lemma `carr_open`: **assumes** `T {is a topology}` **shows** `(⋃T) ∈ T`
  ⟨*proof*⟩

Union of a collection of open sets is open.

lemma `union_open:` **assumes** `T {is a topology}` **and** $\forall$`A`$\in\mathcal{A}$`. A` $\in$ `T`
  shows `(`$\bigcup\mathcal{A}$`)` $\in$ `T` ⟨*proof*⟩

Union of a indexed family of open sets is open.

lemma `union_indexed_open:` **assumes** `A1: T {is a topology}` **and** `A2:` $\forall$`i`$\in$`I. P(i)` $\in$ `T`
  shows `(`$\bigcup$`i`$\in$`I. P(i))` $\in$ `T` ⟨*proof*⟩

The intersection of any nonempty collection of topologies on a set X is a topology.

lemma `Inter_tops_is_top:`
  assumes `A1:` $\mathcal{M} \neq 0$ **and** `A2:` $\forall$`T`$\in\mathcal{M}$`. T {is a topology}`
  shows `(`$\bigcap\mathcal{M}$`) {is a topology}`
⟨*proof*⟩

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`$\in\bigcup$`T` **shows** `{x} {is compact in} T`
  ⟨*proof*⟩

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)` $\equiv$ `Interior(A,T)`

  fixes `cl`
  defines `cl_def [simp]: cl(A)` $\equiv$ `Closure(A,T)`

  fixes `boundary (∂_ [91] 92)`
  defines `boundary_def [simp]: ∂A` $\equiv$ `Boundary(A,T)`

Intersection of a finite nonempty collection of open sets is open.

lemma `(`**in** `topology0) fin_inter_open_open:` **assumes** `N`$\neq$`0 N` $\in$ `FinPow(T)`
  shows $\bigcap$`N` $\in$ `T`
  ⟨*proof*⟩

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⟩
```

62.2 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
  ⟨proof⟩
```

Interior is open.

```
lemma (in topology0) Top_2_L2: shows int(A) ∈ T
⟨proof⟩
```

A set is open iff it is equal to its interior.

```
lemma (in topology0) Top_2_L3: shows U∈T ⟷ int(U) = U
⟨proof⟩
```

Interior of the interior is the interior.

```
lemma (in topology0) Top_2_L4: shows int(int(A)) = int(A)
⟨proof⟩
```

Interior of a bigger set is bigger.

```
lemma (in topology0) interior_mono:
  assumes A1: A⊆B shows int(A) ⊆ int(B)
⟨proof⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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: $V \in T$
  shows $\forall x\in V.\ \exists U\in T.\ (x\in U \wedge U\subseteq V)$
⟨*proof*⟩

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: $\forall x\in V.\ \exists U\in T.\ (x\in U \wedge U\subseteq V)$
  shows $V\in T$
⟨*proof*⟩

The intersection of interiors is a equal to the interior of intersections.

lemma (**in** topology0) int_inter_int: **shows** $\text{int}(A) \cap \text{int}(B) = \text{int}(A\cap B)$
⟨*proof*⟩

62.3 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** $(\bigcup T)$ {is closed in} T
⟨*proof*⟩

Empty set is closed.

lemma (**in** topology0) Top_3_L2: **shows** 0 {is closed in} T
⟨*proof*⟩

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: $A \subseteq \bigcup T$ **shows** $\text{ClosedCovers}(A,T) \neq 0$
⟨*proof*⟩

Intersection of a nonempty family of closed sets is closed.

lemma (**in** topology0) Top_3_L4: **assumes** A1: $K\neq 0$ **and**
  A2: $\forall D\in K.$ D {is closed in} T
  shows $(\bigcap K)$ {is closed in} T
⟨*proof*⟩

The union and intersection of two closed sets are closed.

lemma (**in** topology0) Top_3_L5:
  assumes A1: D_1 {is closed in} T D_2 {is closed in} T
  shows
  $(D_1\cap D_2)$ {is closed in} T

```
  (D₁∪D₂) {is closed in} T
```
⟨*proof*⟩

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*⟩

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
```
⟨*proof*⟩

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*⟩

Boundary of a set is closed.

```
lemma (in topology0) boundary_closed:
  assumes A1: A ⊆ ⋃T shows ∂A {is closed in} T
```
⟨*proof*⟩

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*⟩

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*⟩

A set is contained in its closure.

```
lemma (in topology0) cl_contains_set: assumes A ⊆ ⋃T shows A ⊆ cl(A)
```
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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 ⟨proof⟩
```

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*⟩

end

63 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.

63.1 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*⟩

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*⟩

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 T`$_1$`}` **shows** `T {is T`$_0$`}`
⟨*proof*⟩

If a topology is T_2 then it is T_1.

lemma `T2_is_T1`: **assumes** `A1: T {is T`$_2$`}` **shows** `T {is T`$_1$`}`
⟨*proof*⟩

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 T`$_0$`}` **and** `A2: x ∈ ⋃T  y ∈ ⋃T`
  and `A3: ∀U∈T. (x∈U ⟷ y∈U)`
  shows `x=y`
⟨*proof*⟩

63.2 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⟩
```

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
  ⟨proof⟩
```

Elements of base are open.

```
lemma base_sets_open:
  assumes B {is a base for} T and U ∈ B
  shows U ∈ T
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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: U_1 ∈ {⋃A. A∈Pow(B)} U_2 ∈ {⋃A. A∈Pow(B)}
  and A2: B {satisfies the base condition}
  shows ∃C. C ⊆ {⋃A. A∈Pow(B)} ∧ U_1∩U_2 = ⋃C
⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

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*⟩

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 ⟨*proof*⟩

63.3 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

63.4 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⟩
```

end

64 Metric spaces

```
theory MetricSpace_ZF imports Topology_ZF_1 OrderedLoop_ZF Lattice_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).

64.1 Pseudometric - definition and basic properties

A metric on X is usually defined as a function $d : X \times X \to [0, \infty)$ 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)}
```

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.

```
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)
```

The next lemma shows the definition of the pseudometric in the notation used in the `metric_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⟩
  ⟨proof⟩
```

The values of the metric are 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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

A technical lemma that allows us to shorten some proofs:

```
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⟩
```

If a point x is inside a disk B and $m \leq R - d(c,x)$ 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*⟩

If we assume that the order on the group makes the positive set a meet semi-lattice (i.e. every two-element subset of G_+ has a greatest lower bound) then the collection of disks centered at points of the space and with radii in the positive set of the group satisfies the base condition. The meet semi-lattice assumption can be weakened to "each two-element subset of G_+ has a lower bound in G_+", but we don't do that here.

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*⟩

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 r_x+r_y ≤ d⟨x,y⟩
  shows disk(x,r_x)∩disk(y,r_y) = 0
⟨*proof*⟩

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 r_x<d⟨x,y⟩
  shows (-r_x+(d⟨x,y⟩)) ∈ L_+ **and** disk(x,r_x)∩disk(y,-r_x+(d⟨x,y⟩)) = 0
⟨*proof*⟩

Unions of disks form a topology, hence (pseudo)metric spaces are topological spaces.

theorem (in pmetric_space) pmetric_is_top:
  assumes r {down-directs} L_+
  defines B ≡ ⋃c∈X. {disk(c,R). R∈L_+}
  defines T ≡ {⋃A. A ∈ Pow(B)}
  shows T {is a topology} B {is a base for} T ⋃T = X
⟨*proof*⟩

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)
  ⟨proof⟩
```

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⟩
```

An ordered loop valued metric space is T_2 (i.e. Hausdorff).

```
theorem (in metric_space) metric_space_T2:
    assumes r {down-directs} L+
    defines B ≡ ⋃c∈X. {disk(c,R). R∈L+}
    defines T ≡ {⋃A. A ∈ Pow(B)}
    shows T {is T2}
⟨proof⟩

end
```

65 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.

65.1 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.

```
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 ($\mathbb{R}^+$)
defines nonnegative_def[simp]: $\mathbb{R}^+$ ≡ Nonnegative(ℝ,Add, ROrd)

fixes positiveset ($\mathbb{R}_+$)
defines positiveset_def[simp]: $\mathbb{R}_+$ ≡ PositiveSet(ℝ,Add, ROrd)

fixes setinv (- _ 72)
defines setninv_def [simp]: -A ≡ GroupInv(ℝ,Add)(A)

fixes non_zero ($\mathbb{R}_0$)
defines non_zero_def[simp]: $\mathbb{R}_0$ ≡ 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($\mathbb{R}_0$,restrict(Mul,$\mathbb{R}_0$×$\mathbb{R}_0$))(x)

fixes realsq (_^{2} [96] 97)
defines realsq_def [simp]: x^2 ≡ 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)

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*⟩

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 ROrd
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

Real numbers form an ordered loop.

lemma (**in** reals) reals_loop: **shows** IsAnOrdLoop($\mathbb{R}$,Add,ROrd)
⟨*proof*⟩

The assumptions of the pmetric_space locale hold in the reals locale.

lemma (**in** reals) pmetric_space_valid: **shows** pmetric_space($\mathbb{R}$,Add, ROrd,dist,$\mathbb{R}$)

⟨*proof*⟩

The assumptions of the metric_space locale hold in the reals locale.

lemma (**in** reals) metric_space_valid: **shows** metric_space($\mathbb{R}$,Add, ROrd,dist,$\mathbb{R}$)
⟨*proof*⟩

Some properties of the order relation on reals:

lemma (**in** reals) pos_is_lattice: **shows**
IsLinOrder($\mathbb{R}$,ROrd)
IsLinOrder($\mathbb{R}_+$,ROrd ∩ $\mathbb{R}_+\times\mathbb{R}_+$)
(ROrd ∩ $\mathbb{R}_+\times\mathbb{R}_+$) {is a lattice on} $\mathbb{R}_+$
⟨*proof*⟩

Of course the set of positive real numbers is nonempty as one is there.

lemma (**in** reals) pos_non_empty: **shows** $\mathbb{R}_+\neq 0$
⟨*proof*⟩

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} $\mathbb{R}_+$
⟨*proof*⟩

We define the topology on reals as one consisting of the unions of open disks.

definition (**in** reals) RealTopology ($\tau_{\mathbb{R}}$)
where $\tau_{\mathbb{R}}$ ≡ {⋃A. A ∈ Pow(⋃c∈$\mathbb{R}$.{disk(c,r). r ∈ $\mathbb{R}_+$})}

Real numbers form a Hausdorff topological space with topology generated by open disks.

theorem (**in** reals) reals_is_top:
shows $\tau_{\mathbb{R}}$ {is a topology} ⋃$\tau_{\mathbb{R}}$ = $\mathbb{R}$ $\tau_{\mathbb{R}}$ {is T_2}
⟨*proof*⟩

end

66 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.

66.1 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 ($\mathbf{1}$)
defines one_def[simp]: $\mathbf{1}$ ≡ $\langle\mathbf{1}_R, \mathbf{0}_R\rangle$

fixes zero ($\mathbf{0}$)
defines zero_def[simp]: $\mathbf{0}$ ≡ $\langle\mathbf{0}_R, \mathbf{0}_R\rangle$

fixes iunit (i)
defines iunit_def[simp]: i ≡ $\langle\mathbf{0}_R,\mathbf{1}_R\rangle$

fixes creal ($\mathbb{R}$)
defines creal_def[simp]: $\mathbb{R}$ ≡ {$\langle$r,$\mathbf{0}_R\rangle$. 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 ≡ $\bigcup$ { x ∈ $\mathbb{C}$. b · x = a }

fixes sub (**infixl** - 69)
defines sub_def[simp]: a - b ≡ $\bigcup$ { x ∈ $\mathbb{C}$. b + x = a }

fixes cneg (-_ 95)
defines cneg_def[simp]: - a ≡ $\mathbf{0}$ - a

fixes lessr (**infix** $<_\mathbb{R}$ 68)
defines lessr_def[simp]:
a $<_\mathbb{R}$ b ≡ ⟨a,b⟩ ∈ StrictVersion(CplxROrder(R,A,r))

fixes cpnf ($+\infty$)
defines cpnf_def[simp]: $+\infty$ ≡ $\mathbb{C}$

fixes cmnf ($-\infty$)
defines cmnf_def[simp]: $-\infty$ ≡ {$\mathbb{C}$}

```
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
```

66.2 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*⟩

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*⟩

Real and imaginary parts of sums and products of complex numbers are real.

lemma (**in** complex0) cplx_mul_add_types:
  assumes A1: $z_1 \in \mathbb{C}$ $z_2 \in \mathbb{C}$
  shows
  ReCxAdd(R,A,z_1,z_2) $\in$ R
  ImCxAdd(R,A,z_1,z_2) $\in$ R
  ImCxMul(R,A,M,z_1,z_2) $\in$ R
  ReCxMul(R,A,M,z_1,z_2) $\in$ R
⟨*proof*⟩

Complex reals are complex. Recall the definition of $\mathbb{R}$ in the complex0 locale.

lemma (**in** complex0) axresscn: **shows** $\mathbb{R} \subseteq \mathbb{C}$
  ⟨*proof*⟩

Complex 1 is not complex 0.

lemma (**in** complex0) ax1ne0: **shows** $\mathbf{1} \neq \mathbf{0}$
⟨*proof*⟩

Complex addition is a complex valued binary operation on complex numbers.

lemma (**in** complex0) axaddopr: **shows** CplxAdd(R,A): $\mathbb{C} \times \mathbb{C} \to \mathbb{C}$
⟨*proof*⟩

Complex multiplication is a complex valued binary operation on complex numbers.

lemma (**in** complex0) axmulopr: **shows** CplxMul(R,A,M): $\mathbb{C} \times \mathbb{C} \to \mathbb{C}$
⟨*proof*⟩

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⟩
```

Complex multiplication is commutative.

```
lemma (in complex0) axmulcom: assumes A1: a ∈ ℂ  b ∈ ℂ
  shows a·b = b·a
  ⟨proof⟩
```

A sum of complex numbers is complex.

```
lemma (in complex0) axaddcl: assumes a ∈ ℂ  b ∈ ℂ
  shows a+b ∈ ℂ
  ⟨proof⟩
```

A product of complex numbers is complex.

```
lemma (in complex0) axmulcl: assumes a ∈ ℂ  b ∈ ℂ
  shows  a·b ∈ ℂ
  ⟨proof⟩
```

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⟩
```

Complex addition is commutative.

```
lemma (in complex0) axaddcom: assumes a ∈ ℂ  b ∈ ℂ
  shows a+b = b+a
  ⟨proof⟩
```

Complex addition is associative.

```
lemma (in complex0) axaddass: assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a + b + c = a + (b + c)
⟨proof⟩
```

Complex multiplication is associative.

```
lemma (in complex0) axmulass: assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a · b · c = a · (b · c)
⟨proof⟩
```

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
```

⟨proof⟩

The imaginary unit is a "square root" of -1 (that is, $i^2 + 1 = 0$).

lemma (in complex0) axi2m1: **shows** i·i **+ 1 = 0**
⟨proof⟩

0 is the neutral element of complex addition.

lemma (in complex0) ax0id: **assumes** a ∈ ℂ
shows a **+ 0** = a
⟨proof⟩

The imaginary unit is a complex number.

lemma (in complex0) axicn: **shows** i ∈ ℂ
⟨proof⟩

All complex numbers have additive inverses.

lemma (in complex0) axnegex: **assumes** A1: a ∈ ℂ
shows ∃x∈ℂ. a + x = **0**
⟨proof⟩

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⟩

Complex 1 is a right neutral element for multiplication.

lemma (in complex0) ax1id: **assumes** A1: a ∈ ℂ
shows a·**1** = a
⟨proof⟩

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),$\mathbf{0}_R$⟩
⟨proof⟩

The sum of real numbers is real.

lemma (in complex0) axaddrcl: **assumes** A1: a∈ℝ b∈ℝ
shows a + b ∈ ℝ
⟨proof⟩

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),$\mathbf{0}_R$⟩
⟨proof⟩

The product of (complex) real numbers is real.

lemma (**in** complex0) axmulrcl: **assumes** a∈ℝ b∈ℝ
  shows a · b ∈ ℝ
  ⟨*proof*⟩

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*⟩

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*⟩

Our ℝ symbol is the real axis on the complex plane.

lemma (**in** complex0) real_means_real_axis: **shows** ℝ = ComplexReals(R,A)
  ⟨*proof*⟩

The CplxROrder thing is a relation on the complex reals.

lemma (**in** complex0) cplx_ord_on_cplx_reals:
  shows CplxROrder(R,A,r) ⊆ ℝ×ℝ
  ⟨*proof*⟩

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)) ⊆ ℝ×ℝ
  ⟨*proof*⟩

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∈ℝ b∈ℝ
  ⟨*proof*⟩

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**
  ⟨x,$\mathbf{0}_R$⟩ <$_{\mathbb{R}}$ ⟨y,$\mathbf{0}_R$⟩ ⟷ ⟨x,y⟩ ∈ r ∧ x≠y
⟨*proof*⟩

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*⟩

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<$_\mathbb{R}$b, a=b, b<$_\mathbb{R}$a)
  ⟨*proof*⟩

The strict order on the complex reals is kind of antisymetric.

lemma (**in** complex0) pre_axlttri: **assumes** A1: a ∈ ℝ   b ∈ ℝ
  shows a <$_\mathbb{R}$ b ⟷ ¬(a=b ∨ b <$_\mathbb{R}$ a)
⟨*proof*⟩

The strict order on complex reals is transitive.

lemma (**in** complex0) cplx_strict_ord_trans:
  shows trans(StrictVersion(CplxROrder(R,A,r)))
  ⟨*proof*⟩

The strict order on complex reals is transitive - the explicit version of cplx_strict_ord_trans.

lemma (**in** complex0) pre_axlttrn:
  assumes A1: a <$_\mathbb{R}$ b   b <$_\mathbb{R}$ c
  shows a <$_\mathbb{R}$ c
⟨*proof*⟩

The strict order on complex reals is preserved by translations.

lemma (**in** complex0) pre_axltadd:
  assumes A1: a <$_\mathbb{R}$ b **and** A2: c ∈ ℝ
  shows c+a <$_\mathbb{R}$ c+b
⟨*proof*⟩

The set of positive complex reals is closed with respect to multiplication.

lemma (**in** complex0) pre_axmulgt0: **assumes** A1: **0** <$_\mathbb{R}$ a   **0** <$_\mathbb{R}$ b
  shows **0** <$_\mathbb{R}$ a·b
⟨*proof*⟩

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*⟩

The property of the strict order on complex reals that corresponds to completeness.

lemma (**in** complex0) pre_axsup: **assumes** A1: $X \subseteq \mathbb{R}$ $X \neq 0$ **and**
A2: $\exists x\in\mathbb{R}.\ \forall y\in X.\ y <_{\mathbb{R}} x$
shows
$\exists x\in\mathbb{R}.\ (\forall y\in X.\ \neg(x <_{\mathbb{R}} y)) \wedge (\forall y\in\mathbb{R}.\ (y <_{\mathbb{R}} x \longrightarrow (\exists z\in X.\ y <_{\mathbb{R}} z)))$
⟨*proof*⟩

end

67 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**
$\text{Spec} \equiv \{I\in\mathcal{I}.\ I\triangleleft_p R\}$

The basic set that defines the topoogy is given by the D operator

definition (**in** ring0) openBasic (D) **where**
$S\subseteq R \Longrightarrow D(S) \equiv \{I\in\text{Spec}.\ \neg(S\subseteq I)\}$

The D operator preserves subsets

lemma (**in** ring0) D_operator_preserve_subset:
assumes $S \subseteq T$ $T\subseteq R$
shows $D(S) \subseteq D(T)$
⟨*proof*⟩

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 $T\subseteq R$
shows $D(T) = D(\langle T\rangle_I)$
⟨*proof*⟩

The intersection of to D-sets is the D-set on the product of ideals

lemma (**in** ring0) intersection_open_basic:
assumes $I\triangleleft R$ $J\triangleleft R$
shows $D(I)\cap D(J) = D(I\cdot_I J)$
⟨*proof*⟩

The union of D-sets is the D-set of the sum of the ideals

lemma (**in** ring0) union_open_basic:
  assumes $\mathcal{J} \subseteq \mathcal{I}$
  shows $\bigcup${D(I). I$\in\mathcal{J}$} = D($\oplus_I\mathcal{J}$)
⟨*proof*⟩

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$\in\mathcal{I}$}{is a topology} ⟨*proof*⟩

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$\in\mathcal{I}$})

definition (**in** ring0) ZarCl (cl) **where**
  cl(U) ≡ Closure(U,{D(J). J$\in\mathcal{I}$})

definition (**in** ring0) ZarBound (∂_) **where**
  ∂U ≡ Boundary(U,{D(J). J$\in\mathcal{I}$})

sublocale ring0 < zariski:topology0 {D(J). J$\in\mathcal{I}$}
  ZarInt ZarCl ZarBound ⟨*proof*⟩

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($\bigcap$(Spec-U))
⟨*proof*⟩

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*⟩

corollary (**in** ring0) total_spec:
  shows $\bigcup${D(J). J$\in\mathcal{I}$} = Spec
⟨*proof*⟩

The empty set is the D-set of the zero ideal

lemma (**in** ring0) openBasic_empty:
  shows D({**0**}) = 0
⟨*proof*⟩

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⟩
```

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}
  ⟨proof⟩
```

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·_I K)
⟨proof⟩
```

```
lemma (in ring0) V_intersect:
  assumes 𝒥 ⊆ ℐ 𝒥 ≠0
  shows ⋂{V(I). I∈𝒥} = V(⊕_I 𝒥)
⟨proof⟩
```

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⟩
```

end

68 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 T_0} ⟨proof⟩
```

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}
```

⟨*proof*⟩

end

69 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.

69.1 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*⟩

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 T_2} **and** A2: K {is compact in} T
  shows K {is closed in} T
⟨*proof*⟩

end

70 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.

70.1 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(τ_1,τ_2,f) ≡ (∀U∈τ_2. f-(U) ∈ τ_1)

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(τ_1,τ_2) ≡ {f∈($\bigcup\tau_1$)→($\bigcup\tau_2$). IsContinuous(τ_1,τ_2,f)}

A trivial example of a continuous function - identity is continuous.

lemma id_cont: **shows** IsContinuous(τ,τ,id($\bigcup\tau$))
⟨*proof*⟩

Identity is in the space of continuous functions from $\bigcup \tau$ to itself.

lemma id_cont_sp: **shows** {⟨x,x⟩. x∈$\bigcup\tau$} ∈ Cont(τ,τ)
⟨*proof*⟩

A constant function is continuous.

lemma `const_cont:` **assumes** `T {is a topology}`
  shows `IsContinuous(T,`τ`,ConstantFunction(`$\bigcup$`T,c))`
⟨*proof*⟩

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`$\in\bigcup$`S`
  shows `{`$\langle$`x,c`$\rangle$`. x`$\in\bigcup$`T}` $\in$ `Cont(T,S)`
  ⟨*proof*⟩

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 `cl`$_1$`(A)` and `cl`$_2$`(A)` for closure of a set A in topologies τ_1 and τ_2, respectively.

locale `two_top_spaces0 =`

  fixes τ_1
  assumes `tau1_is_top:` τ_1 `{is a topology}`

  fixes τ_2
  assumes `tau2_is_top:` τ_2 `{is a topology}`

  fixes `X`$_1$
  defines `X1_def [simp]: X`$_1$ $\equiv \bigcup\tau_1$

  fixes `X`$_2$
  defines `X2_def [simp]: X`$_2$ $\equiv \bigcup\tau_2$

  fixes `f`
  assumes `fmapAssum: f: X`$_1$ $\to$ `X`$_2$

  fixes `isContinuous (_ {is continuous} [50] 50)`
  defines `isContinuous_def [simp]: g {is continuous}` $\equiv$ `IsContinuous(`τ_1`,`τ_2`,g)`

  fixes `cl`$_1$
  defines `cl1_def [simp]: cl`$_1$`(A)` $\equiv$ `Closure(A,`τ_1`)`

  fixes `cl`$_2$
  defines `cl2_def [simp]: cl`$_2$`(A)` $\equiv$ `Closure(A,`τ_2`)`

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(`τ_1`)` **and** `topology0(`τ_2`)`
  ⟨*proof*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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} τ_2 **and** A2: ∀U∈B. f-(U) ∈ τ_1
  shows f {is continuous}
⟨*proof*⟩

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: $\mathcal{B}$ {is a base for} τ_1 **and** A2: ∀B∈$\mathcal{B}$. f(B) ∈ τ_2 **and** A3: U∈τ_1
  shows f(U) ∈ τ_2
⟨*proof*⟩

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)
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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} τ_2 {is T_2}
  shows f {is closed in} ProductTopology(τ_1,τ_2)
⟨*proof*⟩

70.2 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))
  ⟨proof⟩
```

Homeomorphisms are open maps.

```
lemma homeo_open: assumes IsAhomeomorphism(T,S,f) and U∈T
  shows f(U) ∈ S
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

70.3 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⟩
```

70.4 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⟩
```

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⟩
```

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⟩
```

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 $g : X_1 \supseteq A \rightarrow X_2$ 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⟩
```

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⟩
```

70.5 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 $f : X_1 \times X_2 \rightarrow X_3$. 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))
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

70.6 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*⟩

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*⟩

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*⟩

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*⟩

0 is continuous.

corollary (**in** `two_top_spaces0`) `zero_continuous:` **shows** `0 {is continuous}`
⟨*proof*⟩

end

71 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⟩

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⟩

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)
  ⟨proof⟩

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⟩
```

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(τt, τo{restricted to}V(ker), g)
⟨proof⟩
```

end

72 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.

72.1 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*⟩

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}`
  ⟨*proof*⟩

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*⟩

72.2 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*⟩

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*⟩

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*⟩

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*⟩

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`
  ⟨*proof*⟩

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*⟩

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*⟩

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⟩
```

end

73 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.

73.1 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⟩
```

Natural numbers are a directed set.

```
corollary Le_directs_nat:
  shows IsLinOrder(nat,Le) Le directs nat
⟨proof⟩
```

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⟩
```

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⟩
```

73.2 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 ≡ (0∉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 0∉𝔉 X∈𝔉 𝔉⊆Pow(X)
    ∀A∈𝔉. ∀B∈𝔉. A∩B∈𝔉 **and** ∀B∈𝔉. ∀C∈Pow(X). B⊆C ⟶ C∈𝔉
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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)

⟨*proof*⟩

A base for a filter determines a filter up to the underlying set.

theorem base_unique_filter:
  assumes C {is a base filter} 𝔉1**and** C {is a base filter} 𝔉2
  shows 𝔉1=𝔉2 ⟷ ⋃𝔉1=⋃𝔉2
⟨*proof*⟩

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*⟩

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}
  ⟨*proof*⟩

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*⟩

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 (_ $\rightarrow_F$ _ 50) **where**
  𝔉{is a filter on}⋃T ⟹ 𝔉$\rightarrow_F$x ≡
  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 $\rightarrow_F$ x
⟨*proof*⟩

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 $\mathfrak{F}$ {is a filter on} $\bigcup$T **and** C {is a base filter} $\mathfrak{F}$ **and** $\mathfrak{F}$ $\rightarrow_F$ x
  shows ∀U∈Pow($\bigcup$T). x∈int(U) ⟶ (∃D∈C. D⊆U) **and** x∈$\bigcup$T
⟨*proof*⟩

A sufficient condition for a filter to converge to a point.

theorem (**in** topology0) convergence_filter_base2:
  assumes $\mathfrak{F}$ {is a filter on} $\bigcup$T **and** C {is a base filter} $\mathfrak{F}$
    and ∀U∈Pow($\bigcup$T). x∈int(U) ⟶ (∃D∈C. D⊆U) **and** x∈$\bigcup$T
  shows $\mathfrak{F}$ $\rightarrow_F$ x
⟨*proof*⟩

A necessary and sufficient condition for a filter to converge to a point.

theorem (**in** topology0) convergence_filter_base_eq:
  assumes $\mathfrak{F}$ {is a filter on} $\bigcup$T **and** C {is a base filter} $\mathfrak{F}$
  shows ($\mathfrak{F}$ $\rightarrow_F$ x) ⟷ ((∀U∈Pow($\bigcup$T). x∈int(U) ⟶ (∃D∈C. D⊆U)) ∧ x∈$\bigcup$T)
⟨*proof*⟩

73.3 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**
  $\mathfrak{F}$ {is a filter on} $\bigcup\mathfrak{F}$ ⟹ Net($\mathfrak{F}$) ≡
    ⟨{⟨A,fst(A)⟩. A∈{⟨x,F⟩∈($\bigcup\mathfrak{F}$)×$\mathfrak{F}$. x∈F}},{⟨A,B⟩∈{⟨x,F⟩∈($\bigcup\mathfrak{F}$)×$\mathfrak{F}$. x∈F}×{⟨x,F⟩∈($\bigcup\mathfrak{F}$)×$\mathfrak{F}$. x∈F}. snd(B)⊆snd(A)}⟩

Net of a filter is indeed a net.

theorem net_of_filter_is_net:
  assumes $\mathfrak{F}$ {is a filter on} X
  shows (Net($\mathfrak{F}$)) {is a net on} X
⟨*proof*⟩

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 $\mathfrak{F}$ {is a filter on} $\bigcup$T **and** $\mathfrak{F}$ $\rightarrow_F$ x
  shows (Net($\mathfrak{F}$)) $\rightarrow_N$ x
⟨*proof*⟩

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

We can use `topology0` locale with filters.

```
lemma topology0_filter:
  assumes 𝔉 {is a filter on} ⋃𝔉
  shows topology0(𝔉 ∪ {0})
  ⟨proof⟩
```

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⟩

end
```

74 Topology and neighborhoods

```
theory Topology_ZF_4a imports Topology_ZF_4
begin
```

This theory considers the relations between topology and systems of neighborhood filters.

74.1 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⟩
```

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⟩
```

74.2 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⟩
```

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⟩
```

74.3 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⟩
```

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)}
  ⟨proof⟩
```

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)}
  ⟨proof⟩
```

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*⟩

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*⟩

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*⟩

74.4 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*⟩

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⟩
```

74.5 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}
  ⟨proof⟩
```

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⟩
```

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)}
  ⟨proof⟩
```

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*⟩

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)`
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

An open set is a set neighborhood of itself.

corollary `open_is_nei:` **assumes** `V∈T`
  shows `V ∈ ({set neighborhood system of} T)(V)`
  ⟨*proof*⟩

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}`
  ⟨*proof*⟩

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*⟩

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*⟩

end

75 Topology - examples

theory `Topology_ZF_examples` **imports** `Topology_ZF Cardinal_ZF`

begin

This theory deals with some concrete examples of topologies.

75.1 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*⟩

We can use theorems proven in `topology0` context for the co-cardinal topology.

```
theorem topology0_CoCardinal:
  assumes InfCard(T)
  shows topology0(CoCardinal(X,T))
  ⟨proof⟩
```

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))
```

75.2 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⟩
```

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⟩
```

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⟩
```

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≠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))` ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

If the cardinal is taken as `T=1` then the topology is indiscrete.

lemma `indiscrete_cocardinal:`
  shows `CoCardinal(X,1) = {0,X}`
⟨*proof*⟩

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*⟩

75.3 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*⟩

We can use `topology0` when discussing excluded set topology.

theorem `topology0_excludedset`:
  shows `topology0(ExcludedSet(X,T))`
  ⟨*proof*⟩

Choosing a singleton set, it is considered a point in excluded topology.

definition
  `ExcludedPoint(X,p) ≡ ExcludedSet(X,{p})`

75.4 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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

75.5 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*⟩

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*⟩

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*⟩

75.6 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*⟩

We can reference the theorems proven in the `topology0` context when discussing the included set topology.

theorem `topology0_includedset:`
  shows `topology0(IncludedSet(X,T))`
  ⟨*proof*⟩

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})
```

75.7 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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

75.8 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⟩
```

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⟩
```

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⟩

end
```

76 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.

76.1 New ideas using a base for a topology

76.2 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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

76.3 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*⟩

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*⟩

76.4 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
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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*⟩

76.5 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*⟩

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)
  ⟨*proof*⟩

lemma topology0_ptopology:
  assumes U {is a partition of} X
  shows topology0(PTopology X U)
  ⟨*proof*⟩

76.6 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
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

76.7 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
  ⟨*proof*⟩

lemma indiscrete_partition:
  assumes X≠0
  shows {X} {is a partition of} X
  ⟨*proof*⟩

theorem discrete_ptopology:
  shows (PTopology X {{x}.x∈X})=Pow(X)
⟨*proof*⟩

theorem indiscrete_ptopology:
  assumes X≠0
  shows (PTopology X {X})={0,X}
⟨*proof*⟩

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*⟩

76.8 Order topologies

76.9 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⟩

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⟩


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⟩

lemma inter_lray_rray:
  assumes bu∈Xcv∈XIsLinOrder(X,r)
  shows LeftRayX(X,r,bu)∩RightRayX(X,r,cv)=IntervalX(X,r,cv,bu)
  ⟨proof⟩

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⟩

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⟩
```

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*⟩

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)
  ⟨*proof*⟩

lemma topology0_ordtopology:
  assumes IsLinOrder(X,r)
  shows topology0(OrdTopology X r)
  ⟨*proof*⟩

76.10 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*⟩

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.

76.11 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.

76.11.1 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*⟩

lemma rightrays_base_condition:
assumes IsLinOrder(X,r)
shows {RightRayX(X,r,b). b∈X}∪{X} {satisfies the base condition}
⟨*proof*⟩

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)
  ⟨*proof*⟩

lemma topology0_lordtopology_rordtopology:
  assumes IsLinOrder(X,r)
  shows topology0(LOrdTopology X r) **and** topology0(ROrdTopology X r)
  ⟨*proof*⟩

76.11.2 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
  ⟨*proof*⟩

76.12 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⟩

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⟩
```

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
  ⟨proof⟩

end
```

77 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.

77.1 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*⟩

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*⟩

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 ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

77.2 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*⟩

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*⟩

77.3 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⟩
```

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⟩
```

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})
  ⟨proof⟩
```

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⟩
```

77.4 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*⟩

In the following proof, we have chosen an infinite cardinal to be able to apply the equation $\mathtt{Q} \times \mathtt{Q} \approx \mathtt{Q}$. 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*⟩

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)))
```
⟨*proof*⟩

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*⟩

theorem `Q_disc_is_second_card_csuccQ:`
```
  assumes InfCard(Q)
  shows Pow(Q){is of second type of cardinal}csucc(Q)
```
⟨*proof*⟩

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))) $\longleftrightarrow$ ({the axiom of}Q{choice holds for subsets}(Pow(Q)))
⟨*proof*⟩

end

78 Topology 5

theory Topology_ZF_5 **imports** Topology_ZF_properties Topology_ZF_examples_1 Topology_ZF_4
begin

78.1 Some results for separation axioms

First we will give a global characterization of T_1-spaces; which is interesting because it involves the cardinal $\mathbb{N}$.

lemma (**in** topology0) T1_cocardinal_coarser:
shows (T {is T_1}) $\longleftrightarrow$ (CoFinite ($\bigcup$T))⊆T
⟨*proof*⟩

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 T_1}) $\longleftrightarrow$ (∀x∈$\bigcup$T. {x}{is closed in}T)
⟨*proof*⟩

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*⟩

corollary cocardinal_is_T1:
fixes X K
assumes InfCard(K)
shows CoCardinal(X,K) {is T_1}
⟨*proof*⟩

In T_2-spaces, filters and nets have at most one limit point.

lemma (**in** topology0) T2_imp_unique_limit_filter:
assumes T {is T_2} $\mathfrak{F}$ {is a filter on}$\bigcup$T $\mathfrak{F}$ $\rightarrow_F$ x $\mathfrak{F}$ $\rightarrow_F$ y
shows x=y

⟨proof⟩

lemma (**in** topology0) T2_imp_unique_limit_net:
  assumes T {is T_2} N {is a net on}⋃T N $\rightarrow_N$ x N $\rightarrow_N$ y
  shows x=y
⟨proof⟩

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 $\rightarrow_F$ x) ∧ (𝔉 $\rightarrow_F$ y)) ⟶ x=y
  shows T {is T_2}
⟨proof⟩

lemma (**in** topology0) unique_limit_net_imp_T2:
  assumes ∀x∈⋃T. ∀y∈⋃T. ∀N. ((N {is a net on}⋃T) ∧ (N $\rightarrow_N$ x) ∧ (N $\rightarrow_N$ y)) ⟶ x=y
  shows T {is T_2}
⟨proof⟩

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 T_2} 𝔉 {is a filter on} ⋃𝔉 x∈⋃𝔉
  shows ⋃𝔉={x}
⟨proof⟩

There are more separation axioms that just T_0, T_1 or T_2

definition
  isT3 (_{is T_3} 90)
  where T{is T_3} ≡ (T{is T_1}) ∧ (T{is regular})

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 T_4} 90)
  where T{is T_4} ≡ (T{is T_1}) ∧ (T{is normal})

lemma (**in** topology0) T4_is_T3:
  assumes T{is T_4} **shows** T{is T_3}
⟨proof⟩

lemma (**in** topology0) T3_is_T2:
  assumes T{is T_3} **shows** T{is T_2}
⟨*proof*⟩

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*⟩

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*⟩

lemma (**in** topology0) regular_eq:
  shows T{is regular} ⟷ (∀x∈⋃T. ∀U∈T. x∈U ⟶ (∃V∈T. x∈V∧ cl(V)⊆U))
  ⟨*proof*⟩

A Hausdorff space separates compact spaces from points.

theorem (**in** topology0) T2_compact_point:
  assumes T{is T_2} A{is compact in}T x∈⋃T x∉A
  shows ∃U∈T. ∃V∈T. A⊆U ∧ x∈V ∧ U∩V=0
⟨*proof*⟩

A Hausdorff space separates compact spaces from other compact spaces.

theorem (**in** topology0) T2_compact_compact:
  assumes T{is T_2} 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*⟩

A compact Hausdorff space is normal.

corollary (**in** topology0) T2_compact_is_normal:
  assumes T{is T_2} (⋃T){is compact in}T
  shows T{is normal} ⟨*proof*⟩

78.2 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⟩

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⟩

corollary here_regular:
  shows IsRegular {is hereditary} ⟨proof⟩

theorem T1_here:
  assumes T{is T1} A∈Pow(⋃T) shows (T{restricted to}A){is T1}
⟨proof⟩

corollary here_T1:
  shows isT1 {is hereditary} ⟨proof⟩

lemma here_and:
  assumes P {is hereditary} Q {is hereditary}
  shows (λT. P(T) ∧ Q(T)) {is hereditary} ⟨proof⟩

corollary here_T3:
  shows isT3 {is hereditary} ⟨proof⟩

lemma T2_here:
  assumes T{is T2} A∈Pow(⋃T) shows (T{restricted to}A){is T2}
⟨proof⟩

corollary here_T2:
  shows isT2 {is hereditary} ⟨proof⟩

lemma T0_here:
  assumes T{is T0} A∈Pow(⋃T) shows (T{restricted to}A){is T0}
⟨proof⟩

corollary here_T0:
  shows isT0 {is hereditary} ⟨proof⟩
```

78.3 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)` $\equiv$ $\forall$`T. ((T{is a topology}` $\wedge$ $\bigcup$`T`$\approx$`K)` $\longrightarrow$ `P(T))`

lemma `equipollent_spect:`
  assumes `A`$\approx$`B B {is in the spectrum of} P`
  shows `A {is in the spectrum of} P`
⟨*proof*⟩

theorem `eqpoll_iff_spec:`
  assumes `A`$\approx$`B`
  shows `(B {is in the spectrum of} P)` $\longleftrightarrow$ `(A {is in the spectrum of} P)`
⟨*proof*⟩

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)` $\longleftrightarrow$ `A` $\lesssim$ `1`
⟨*proof*⟩

If the topological properties are related, then so are the spectra.

lemma `P_imp_Q_spec_inv:`
  assumes $\forall$`T. T{is a topology}` $\longrightarrow$ `(Q(T)` $\longrightarrow$ `P(T))  A {is in the spectrum of} Q`
  shows `A {is in the spectrum of} P`
⟨*proof*⟩

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)` $\longleftrightarrow$ `A` $\lesssim$ `1`
⟨*proof*⟩

theorem `T1_spectrum:`
  shows `(A {is in the spectrum of} isT1)` $\longleftrightarrow$ `A` $\lesssim$ `1`
⟨*proof*⟩

theorem `T2_spectrum:`
  shows `(A {is in the spectrum of} isT2)` $\longleftrightarrow$ `A` $\lesssim$ `1`
⟨*proof*⟩

theorem `T3_spectrum:`

shows (A {is in the spectrum of} isT3) $\longleftrightarrow$ A $\lesssim$ 1
⟨*proof*⟩

theorem compact_spectrum:
shows (A {is in the spectrum of} (λT. ($\bigcup$T) {is compact in}T)) $\longleftrightarrow$ Finite(A)
⟨*proof*⟩

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. (($\bigcup$T){is compact of cardinal} csucc(K){in}T))) $\longleftrightarrow$ (A$\lesssim$K)
⟨*proof*⟩

theorem compactK_spectrum_reverse:
assumes $\forall$A. (A {is in the spectrum of} (λT. (($\bigcup$T){is compact of cardinal} csucc(K){in}T))) $\longleftrightarrow$ (A$\lesssim$K) InfCard(K)
shows {the axiom of}K{choice holds for subsets}(Pow(K))
⟨*proof*⟩

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 $\equiv$ $\forall$A$\in$Pow($\bigcup$T). P(T{restricted to}A) $\longrightarrow$ (A {is in the spectrum of} P)

abbreviation
ANTI(P) $\equiv$ 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 $\forall$T. T{is a topology} $\longrightarrow$ P(T) $\longrightarrow$ Q(T) $\forall$A. (A{is in the spectrum of}Q) $\longrightarrow$ (A{is in the spectrum of}P)
and T{is anti-}Q
shows T{is anti-}P
⟨*proof*⟩

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⟩
```

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⟩
```

The anti-properties are always hereditary

```
theorem anti_here:
  shows ANTI(P){is hereditary}
⟨proof⟩

corollary (in topology0) anti_imp_anti3:
  assumes T{is anti-}P
  shows T{is anti-}ANTI(ANTI(P))
  ⟨proof⟩
```

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⟩

lemma indiscrete_spectrum:
  shows (A {is in the spectrum of}(λT. T={0,⋃T})) ⟷ A≲1
⟨proof⟩

theorem (in topology0) anti_indiscrete:
  shows (T{is anti-}(λT. T={0,⋃T})) ⟷ T{is T₀}
⟨proof⟩
```

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⟩
```

```
corollary linordtop_here:
  shows (λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})){is hereditary}
  ⟨proof⟩
```

```
theorem (in topology0) anti_T4:
  shows (T{is anti-}isT4) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
⟨proof⟩
```

```
theorem (in topology0) anti_T3:
  shows (T{is anti-}isT3) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
⟨proof⟩
```

```
theorem (in topology0) anti_T2:
  shows (T{is anti-}isT2) ⟷ (IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))
⟨proof⟩
```

```
lemma linord_spectrum:
  shows (A{is in the spectrum of}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T).
U⊆V}))) ⟷ A≲1
⟨proof⟩
```

```
theorem (in topology0) anti_linord:
  shows (T{is anti-}(λT. IsLinOrder(T,{⟨U,V⟩∈Pow(⋃T)×Pow(⋃T). U⊆V})))
⟷ T{is T₁}
⟨proof⟩
```

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

79 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.

79.1 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*⟩

79.2 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*⟩

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*⟩

79.3 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 $\mathfrak{F}$ {is a filter on} X_1 f {is continuous} $\mathfrak{F} \rightarrow_F$ x {in} τ_1
  shows (f[$\mathfrak{F}$]..X_2) $\rightarrow_F$ (f(x)) {in} τ_2
⟨*proof*⟩

Continuity in filter at every point of the domain implies global continuity.

lemma (in two_top_spaces0) filter_conver_preserved_imp_cont:
  assumes $\forall$x$\in\bigcup\tau_1$. $\forall\mathfrak{F}$. (($\mathfrak{F}$ {is a filter on} X_1) $\wedge$ ($\mathfrak{F} \rightarrow_F$ x {in} τ_1))
$\longrightarrow$ ((f[$\mathfrak{F}$]..X_2) $\rightarrow_F$ (fx) {in} τ_2)
  shows f{is continuous}
⟨*proof*⟩

end

80 Topology 7

theory Topology_ZF_7 **imports** Topology_ZF_5
begin

80.1 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} $\equiv$ $\forall$U. (U$\in$T $\wedge$ (U {is closed in}T)) $\longrightarrow$ U=0$\vee$U=$\bigcup$T

lemma indiscrete_connected:
  shows {0,X} {is connected}
  ⟨*proof*⟩

The anti-property of connectedness is called total-diconnectedness.

definition IsTotDis (_ {is totally-disconnected} 70)
  where IsTotDis $\equiv$ ANTI(IsConnected)

lemma conn_spectrum:
  shows (A{is in the spectrum of}IsConnected) $\longleftrightarrow$ A$\lesssim$1
⟨*proof*⟩

The discrete space is a first example of totally-disconnected space.

```
lemma discrete_tot_dis:
  shows Pow(X) {is totally-disconnected}
⟨proof⟩
```

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⟩
```

```
lemma Indiscrete_HConn:
  shows {0,X}{is hyperconnected}
  ⟨proof⟩
```

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⟩
```

```
lemma HConn_spectrum:
  shows (A{is in the spectrum of}IsHConnected) ⟷ A≲1
⟨proof⟩
```

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⟩
```

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⟩
```

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*⟩

Every sober space is T_0.

lemma (**in** topology0) sober_imp_T0:
  assumes T{is sober}
  shows T{is T_0}
⟨*proof*⟩

Every T_2 space is anti-hyperconnected.

theorem (**in** topology0) T2_imp_anti_HConn:
  assumes T{is T_2}
  shows T{is anti-}IsHConnected
⟨*proof*⟩

Every anti-hyperconnected space is T_1.

theorem anti_HConn_imp_T1:
  assumes T{is anti-}IsHConnected
  shows T{is T_1}
⟨*proof*⟩

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 T_1}
⟨*proof*⟩

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*⟩

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 T₂})` **and**
    `(joinT {CoFinite nat,ExcludedSet(nat,{0,1})}){is anti-} IsHConnected`
⟨*proof*⟩

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*⟩

theorem (**in** `topology0) anti_HConn_iff_T1_sober:`
  shows `(T{is anti-}IsHConnected) ⟷ (T{is sober}∧T{is T₁})`
  ⟨*proof*⟩

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*⟩

Every ultraconnected space is connected.

lemma `UConn_imp_Conn:`
  assumes `T{is ultraconnected}`
  shows `T{is connected}`
⟨*proof*⟩

lemma `UConn_spectrum:`
  shows `(A{is in the spectrum of}IsUConnected) ⟷ A≲1`
⟨*proof*⟩

This time, anti-ultraconnected is an old property.

theorem (**in** `topology0) anti_UConn:`
  shows `(T{is anti-}IsUConnected) ⟷ T{is T₁}`
⟨*proof*⟩

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⟩

theorem (in topology0)anti_sober:
  shows (T{is anti-}IsSober) ⟷ T={0,⋃T}
⟨proof⟩


end
```

81 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.

81.1 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⟩
```

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)
    ⟨proof⟩
```

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⟩
```

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⟩
```

81.2 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 ⟨proof⟩
```

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*⟩

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⟩
```

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⟩
```

We apply previous results to this topology.

```
theorem (in topology0) total_quo_equi:
  assumes equiv(⋃T,r)
  shows ⋃({quotient by}r)=(⋃T)//r
  ⟨proof⟩
```

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}
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩

end
```

82 Topology 9

```
theory Topology_ZF_9
imports Topology_ZF_2 Group_ZF_2 Topology_ZF_7 Topology_ZF_8
begin
```

82.1 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*⟩

The identity function is a homeomorphism.

lemma (**in** topology0) homeo_id:
  shows id(⋃T)∈HomeoG(T)
⟨*proof*⟩

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*⟩

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*⟩

82.2 Examples computed

As a first example, we show that the group of homeomorphisms of the co-cardinal topology is the group of bijective functions.

theorem homeo_cocardinal:
  assumes InfCard(Q)
  shows HomeoG(CoCardinal(X,Q))=bij(X,X)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

This last example shows that order isomorphic sets give homeomorphic topological spaces.

82.3 Properties preserved by functions

The continuous image of a connected space is connected.

theorem (**in** `two_top_spaces0`) `cont_image_conn`:
  assumes `IsContinuous(`τ_1`,`τ_2`,f) f∈surj(`X_1`,`X_2`)` τ_1`{is connected}`
  shows τ_2`{is connected}`
⟨*proof*⟩

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(`τ_1`,`τ_2`,f)` τ_1`{is connected}` τ_2`{is totally-disconnected}`
`f:`X_1`→`X_2 X_1`≠0`
  shows `∃q∈`X_2`. ∀w∈`X_1`. f(w)=q`
⟨*proof*⟩

The continuous image of a compact space is compact.

theorem (**in** `two_top_spaces0`) `cont_image_com`:
  assumes `IsContinuous(`τ_1`,`τ_2`,f) f∈surj(`X_1`,`X_2`)` X_1`{is compact of cardinal}K{in}`τ_1

```
  shows X₂{is compact of cardinal}K{in}τ₂
⟨proof⟩
```

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⟩
```

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⟩

corollary(in topology0) ConnQuot:
  assumes T{is connected} equiv(⋃T,r)
  shows ({quotient by}r){is connected}
⟨proof⟩

end
```

83 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.

83.1 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⟩
```

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⟩
```

83.2 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⟩
```

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⟩
```

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⟩
```

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⟩
```

83.3 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⟩
```

```
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⟩
```

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⟩
```

end

84 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`.

84.1 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.

84.2 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⟩
```

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.

84.3 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)
```
  ⟨*proof*⟩

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)
```
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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⟩
```

```
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⟩
```

84.4 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⟩
```

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⟩
```

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⟩
```

```
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⟩
```

end

85 Properties in topology 2

```
theory Topology_ZF_properties_2 imports Topology_ZF_7 Topology_ZF_1b
  Finite_ZF_1 Topology_ZF_11
```

begin

85.1 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)))
```

85.2 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⟩
```

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⟩
```

85.3 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) ⟨proof⟩
```

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 ⟨proof⟩
```

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⟩
```

Every set, with the cofinite topology is compact.

```
lemma cofinite_compact:
  shows X {is compact in}(CoFinite X) ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

85.4 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} ⟨*proof*⟩

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*⟩

We added only one new point to the space.

lemma (**in** topology0) op_compact_total:
  shows ⋃({one-point compactification of}T)={⋃T}∪(⋃T)
⟨*proof*⟩

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)
⟨*proof*⟩

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 T_2}
  shows T{is T_2}
  ⟨*proof*⟩

lemma (**in** topology0) op_compact_T2_2:
  assumes ({one-point compactification of}T){is T_2}
  shows T{is locally-compact}
⟨*proof*⟩

lemma (**in** topology0) op_compact_T2_3:
  assumes T{is locally-compact} T{is T_2}
  shows ({one-point compactification of}T){is T_2}
⟨*proof*⟩

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 T_2}
  shows T{is regular}

⟨*proof*⟩

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*⟩

85.5 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*⟩

```
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-T2}) ⟷ (∀x∈⋃T. ∃A∈T. x∈A∧(T{restricted to}A){is
T2})
```
⟨*proof*⟩

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⟩
```

```
lemma(in topology0) locally_T2_is_antiHConn:
  assumes T{is locally-T2}
  shows T{is anti-}IsHConnected
⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

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⟩
```

There can be considered many more local properties, which; as happens with locally-T_2; can distinguish between spaces other properties cannot.

end

86 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.

86.1 More anti-properties

In this section we study more anti-properties.

86.2 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⟩

theorem pow_anti_compact:
  shows Pow(A){is anti-compact}
⟨proof⟩
```

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.

86.3 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)
  ⟨proof⟩

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⟩
```

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⟩
```

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⟩

theorem op_comp_pow_nat_no_anti_comp:
  shows ¬(({one-point compactification of}Pow(nat)){is anti-compact})
⟨proof⟩
```

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 $\mathbb{N}$.

```
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})
  ⟨proof⟩

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}))
  ⟨proof⟩
```

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⟩
```

This result is interesting because depending on the different axioms we add to ZF, it means two different things:

- Every subspace of $\mathbb{N}$ is Lindeloef.
- Only the compact subspaces of $\mathbb{N}$ 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*⟩

lemma countable_lord_nat:
  shows {LeftRayX(nat,Le,n). n∈nat} ∪{nat} ∪{0}≺csucc(nat)
⟨*proof*⟩

corollary lindelof_lord_nat:
  shows nat{is lindeloef in}(LOrdTopology nat Le)
  ⟨*proof*⟩

theorem not_comp_lord_nat:
  shows ¬(nat{is compact in}(LOrdTopology nat Le))
⟨*proof*⟩

86.4 More Separation properties

In this section we study more separation properties.

86.5 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

86.6 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 T_2}
  shows T{is KC}
⟨*proof*⟩

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 T_1}
⟨*proof*⟩

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*⟩

US spaces are also T_1.

theorem (**in** topology0) US_imp_T1:
  assumes T{is US}
  shows T{is T_1}
⟨*proof*⟩

86.7 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*⟩

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*⟩

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⟩
```

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⟩
```

The cocountable topology on $\mathbb{N}^+$ is not anti-hyperconnected.

```
corollary cocountable_in_csucc_nat_notAntiHConn:
  shows ¬((CoCountable csucc(nat)){is anti-}IsHConnected)
⟨proof⟩
```

The cocountable topology on $\mathbb{N}^+$ is not T_2.

```
theorem cocountable_in_csucc_nat_noT2:
  shows ¬(CoCountable csucc(nat)){is T₂}
⟨proof⟩
```

The cocountable topology on $\mathbb{N}^+$ is T_1.

```
theorem cocountable_in_csucc_nat_T1:
  shows (CoCountable csucc(nat)){is T₁}
  ⟨proof⟩
```

The cocountable topology on $\mathbb{N}^+$ is anti-compact.

```
theorem cocountable_in_csucc_nat_antiCompact:
  shows (CoCountable csucc(nat)){is anti-compact}
⟨proof⟩
```

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*⟩

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*⟩

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*⟩

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*⟩

This topology is Hausdorff.

theorem extension_pow_T2:
  shows (Pow(csucc(nat)) ∪ {{csucc(nat)}∪S. S∈(CoCountable csucc(nat))-{0}}){is T_2}
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

In conclusion, $US \nRightarrow KC$.

86.8 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.

86.9 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})
```

86.10 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*⟩

The discrete space is clearly scattered:

lemma pow_is_scattered:
  shows Pow(A){is scattered}
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

86.11 Structural results

Every resolvable space is also perfect.

theorem (**in** `topology0`) `resolvable_imp_perfect:`
  assumes `T{is resolvable}`
  shows `T{is perfect}`
⟨*proof*⟩

The spectrum of being resolvable follows:

corollary `spectrum_resolvable:`
  shows `(A{is in the spectrum of}IsRes) ⟷ A=0`
⟨*proof*⟩

The cofinite space over $\mathbb{N}$ is a T_1, perfect and luzin space.

theorem `cofinite_nat_perfect:`
  shows `(CoFinite nat){is perfect}`
⟨*proof*⟩

theorem `cofinite_nat_luzin:`
  shows `(CoFinite nat){is luzin}`
⟨*proof*⟩

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*⟩

theorem `cocountable_csucc_nat_luzin:`
  shows `(CoCountable csucc(nat)){is luzin}`
⟨*proof*⟩

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

87 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.

87.1 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 pmetric_space1 locale extends the reals locale, adding the carrier X of the metric space and the metric $\mathfrak{d}$ to the context, together with the assumption that $\mathfrak{d} : X \times X \to \mathbb{R}$ is a pseudo metric. An alternative would be to define the pmetric_space1 as an extension of the pmetric_space1 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 pmetric_space1 = 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)
```

The propositions proven in the pmetric_space context defined in Metric_Space_ZF theory are valid in the pmetric_space1 context.

```
lemma (in pmetric_space1) pmetric_space_pmetric_space1_valid:
  shows pmetric_space(ℝ,Add,ROrd,d,X)
  ⟨proof⟩
```

It is convenient to have the collection of all open balls in given (p) metrics defined as a separate notion.

```
definition (in pmetric_space1) Open_Balls
  where Open_Balls ≡ ⋃c∈X. {ball(c,r). r ∈ ℝ₊}
```

Topology on a metric space is defined as the collection of sets that are unions of open balls of the (p)metric.

```
definition (in pmetric_space1) Metric_Topology
```

```
  where Metric_Topology ≡ {⋃A. A ∈ Pow(Open_Balls)}
```

The `metric_space1` locale (context) specializes the the `pmetric_space1` context by adding the assumption of identity of indiscernibles.

```
locale metric_space1 = pmetric_space1 +
  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 `metric_space1` context.

```
lemma (in metric_space1) metric_space_metric_space1_valid:
  shows metric_space(ℝ,Add,ROrd,d,X)
  ⟨proof⟩
```

87.2 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.

Since in the `pmetric_space1` context 𝔡 is a pseudometrics the (p)metric topology as defined above is indeed a topology, the set of open balls is the base of that topology and the carrier of the topology is the underlying (p)metric space carrier X.

```
theorem (in pmetric_space1) rpmetric_is_top:
  shows
    Metric_Topology {is a topology}
    Open_Balls {is a base for} Metric_Topology
    ⋃ Metric_Topology = X
  ⟨proof⟩
```

The topology generated by a metric is Hausdorff (i.e. T_2).

```
theorem (in metric_space1) rmetric_space_T2: shows Metric_Topology {is
T₂}
  ⟨proof⟩

end
```

88 Uniform spaces

```
theory UniformSpace_ZF imports Topology_ZF_2 Topology_ZF_4a
begin
```

This theory defines uniform spaces and proves their basic properties.

88.1 Definition and motivation

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)
```

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} ≠ 0 and x ∈ W{x}
⟨proof⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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*⟩

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*⟩

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*⟩

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`
  ⟨*proof*⟩

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*⟩

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⟩
```

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
  ⟨proof⟩
```

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}
  ⟨proof⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

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 $X \ni x \mapsto \{V\{x\}|x \in X\}$ (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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

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} ⟨proof⟩
```

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⟩
```

Uniform spaces are regular (T_3).

```
theorem utopreg:
  assumes Φ {is a uniformity on} X
  shows UniformTopology(Φ,X) {is regular}
⟨proof⟩

end
```

89 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.

89.1 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*⟩

end

90 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 an alternative definition based of the notion of uniform covers.

90.1 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`
  ⟨*proof*⟩

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)`
  ⟨*proof*⟩

An alternative formula for star of a singleton.

lemma `star_singleton:` **shows** `(⋃{V×V. V∈P}){x} = Star({x},P)`
  ⟨*proof*⟩

Star of a larger set is larger.

lemma `star_mono`: **assumes** `U⊆V` **shows** `Star(U,P) ⊆ Star(V,P)`
⟨*proof*⟩

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)`
⟨*proof*⟩

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}`
⟨*proof*⟩

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}`
⟨*proof*⟩

Star refinement implies barycentric refinement.

lemma `star_is_bary`: **assumes** `Q∈Covers(X)` **and** `Q <`*` P`
shows `Q <`^{B}` P`
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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)`
  ⟨*proof*⟩

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 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*⟩

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*⟩

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}
```
⟨*proof*⟩

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}
```
⟨*proof*⟩

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*⟩

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)
⟨*proof*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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*⟩

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*⟩

end

91 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.

91.1 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]: ∑k ≡ Fold1(f,k)
```

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)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

We can use the group0 locale in the context of topgroup.

```
lemma (in topgroup) group0_valid_in_tgroup: shows group0(G,f)
  ⟨proof⟩
```

We can use the group0 locale in the context of topgroup.

```
sublocale topgroup < group0 G f gzero grop grinv
    ⟨proof⟩
```

We can use semigr0 locale in the context of topgroup.

```
lemma (in topgroup) semigr0_valid_in_tgroup: shows semigr0(G,f)
  ⟨proof⟩
```

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)
  ⟨proof⟩
```

Negative of a group element is in group.

```
lemma (in topgroup) neg_in_tgroup: assumes g∈G shows (-g) ∈ G
  ⟨proof⟩
```

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
  ⟨proof⟩
```

Zero is in the group.

```
lemma (in topgroup) zero_in_tgroup: shows 0∈G
  ⟨proof⟩
```

Another lemma about canceling with two group elements written in additive notation

lemma (in topgroup) inv_cancel_two_add:
  assumes $x_1 \in G$ $x_2 \in G$
  shows
    $x_1+(-x_2)+x_2 = x_1$
    $x_1+x_2+(-x_2) = x_1$
    $(-x_1)+(x_1+x_2) = x_2$
    $x_1+((-x_1)+x_2) = x_2$
  ⟨*proof*⟩

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** $x_1 \in G$ $x_2 \in G$ $x_3 \in G$
  shows
    $(x_1+(-x_2))+(x_2+(-x_3)) = x_1+ (-x_3)$
    $((-x_1)+x_2)+((-x_2)+x_3) = (-x_1)+ x_3$
    $(- (x_1+x_2)) + (x_1+x_3) = (-x_2)+x_3$
    $(x_1+x_2) + (-(x_3+x_2)) =x_1+ (-x_3)$
    $(-x_1) + (x_1+x_2+x_3) + (-x_3) = x_2$
⟨*proof*⟩

We can cancel an element on the right from both sides of an equation.

lemma (in topgroup) cancel_right_add:
  assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $x_1+x_2 = x_3+x_2$
  shows $x_1 = x_3$
  ⟨*proof*⟩

We can cancel an element on the left from both sides of an equation.

lemma (in topgroup) cancel_left_add:
  assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $x_1+x_2 = x_1+x_3$
  shows $x_2 = x_3$
  ⟨*proof*⟩

We can put an element on the other side of an equation.

lemma (in topgroup) put_on_the_other_side:
  assumes $x_1 \in G$ $x_2 \in G$ $x_3 = x_1+x_2$
  shows $x_3+(-x_2) = x_1$ **and** $(-x_1)+x_3 = x_2$
  ⟨*proof*⟩

A simple equation from lemma simple_equation0 in Group_ZF in additive notation

lemma (in topgroup) simple_equation0_add:
  assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $x_1+(-x_2) = (-x_3)$
  shows $x_3 = x_2 + (-x_1)$
  ⟨*proof*⟩

A simple equation from lemma simple_equation1 in Group_ZF in additive notation

lemma (**in** topgroup) simple_equation1_add:
  assumes $x_1 \in G$ $x_2 \in G$ $x_3 \in G$ $(-x_1)+x_2 = (-x_3)$
  shows $x_3 = (-x_2) + x_1$
  ⟨*proof*⟩

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}
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

91.2 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*⟩

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⟩
```

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}
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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
  ⟨proof⟩
```

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))
⟨proof⟩
```

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
  ⟨proof⟩
```

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))
  ⟨proof⟩
```

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⟩
```

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⟩
```

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
  ⟨proof⟩
```

Inverse of an open set is open.

```
lemma (in topgroup) open_inv_open: assumes V∈T shows (-V) ∈ T
  ⟨proof⟩
```

Inverse is a homeomorphism.

```
lemma (in topgroup) inv_homeo: shows IsAhomeomorphism(T,T,GroupInv(G,f))
  ⟨proof⟩
```

Taking negative preserves interior.

lemma (**in** topgroup) int_inv_inv_int: **assumes** $A \subseteq G$
  shows int(-A) = -(int(A))
  ⟨*proof*⟩

91.3 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** $G \in \mathcal{N}_0$
  ⟨*proof*⟩

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** $A \subseteq G$ $g \in A$ $H \in \mathcal{N}_0$
  shows $g \in$ A+int(H)
⟨*proof*⟩

Any element belongs to the interior of any neighboorhood of zero left translated by that element.

lemma (**in** topgroup) elem_in_int_ltrans:
  assumes $g \in G$ **and** $H \in \mathcal{N}_0$
  shows $g \in$ int(g+H) **and** $g \in$ int(g+H) + int(H)
⟨*proof*⟩

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: $g \in G$ **and** A2: $H \in \mathcal{N}_0$
  shows $g \in$ int(H+g) **and** $g \in$ int(H+g) + int(H)
⟨*proof*⟩

Negative of a neighborhood of zero is a neighborhood of zero.

lemma (**in** topgroup) neg_neigh_neigh: **assumes** $H \in \mathcal{N}_0$
  shows $(-H) \in \mathcal{N}_0$
⟨*proof*⟩

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: $U \in T$ **and** $g \in U$
  shows $(-g)+U \in \mathcal{N}_0$
⟨*proof*⟩

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) ∈ $\mathcal{N}_0$
⟨*proof*⟩

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∈$\mathcal{N}_0$ **and** x∈G
  shows x+(W+(-x)) ∈ $\mathcal{N}_0$ **and** (x+W)+(-x) ∈ $\mathcal{N}_0$
⟨*proof*⟩

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*⟩

Every neighborhood of zero has a symmetric subset that is a neighborhood of zero.

theorem (**in** topgroup) exists_sym_zerohood:
  assumes U∈$\mathcal{N}_0$
  shows ∃V∈$\mathcal{N}_0$. (V⊆U ∧ (-V)=V)
⟨*proof*⟩

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∈$\mathcal{N}_0$
  shows ∃V∈$\mathcal{N}_0$. (V⊆U∧ (V+V)⊆U ∧ (-V)=V)
⟨*proof*⟩

91.4 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 ∈ $\mathcal{N}_0$
  shows cl(A) ⊆ A+H
⟨*proof*⟩

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∈$\mathcal{N}_0$. A+H)
⟨*proof*⟩

91.5 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
⟨*proof*⟩

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 ⟨*proof*⟩

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*⟩

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*⟩
end

92 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.

92.1 Separation properties of topological groups

The topological groups have very specific properties. For instance, G is $\mathtt{T}_0$ iff it is $\mathtt{T}_3$.

theorem(**in** topgroup) cl_point:
  assumes x∈G
  shows cl({x}) = (⋂H∈$\mathcal{N}_0$. x+H)

⟨*proof*⟩

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*⟩

```
theorem (in topgroup) T0_imp_neu_closed:
  assumes T{is T₀}
  shows {0}{is closed in}T
```
⟨*proof*⟩

92.2 Existence of nice neighbourhoods.

```
lemma (in topgroup) exist_basehoods_closed:
  assumes U∈𝒩₀
  shows ∃V∈𝒩₀. cl(V)⊆U
```
⟨*proof*⟩

92.3 Rest of separation axioms

```
theorem(in topgroup) T1_imp_T2:
  assumes T{is T₁}
  shows T{is T₂}
```
⟨*proof*⟩

Here follow some auxiliary lemmas.

```
lemma (in topgroup) trans_closure:
  assumes x∈G A⊆G
  shows cl(x+A)=x+cl(A)
```
⟨*proof*⟩

```
lemma (in topgroup) trans_interior2: assumes A1: g∈G and A2: A⊆G
  shows int(A)+g = int(A+g)
```
⟨*proof*⟩

```
lemma (in topgroup) trans_closure2:
  assumes x∈G A⊆G
  shows cl(A+x)=cl(A)+x
```
⟨*proof*⟩

```
lemma (in topgroup) trans_subset:
  assumes A⊆((-x)+B)x∈GA⊆GB⊆G
  shows x+A⊆B
```
⟨*proof*⟩

Every topological group is regular, and hence T_3. The proof is in the next section, since it uses local properties.

92.4 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∩$\mathcal{N}_0$. ∃N∈$\mathcal{N}_0$. N⊆U∧ P(N,T) ∀N∈Pow(G). ∀x∈G. P(N,T) ⟶ P(x+N,T)
  shows T{is locally}P
⟨*proof*⟩

lemma (**in** topgroup) trans_closed:
  assumes A{is closed in}Tx∈G
  shows (x+A){is closed in}T
⟨*proof*⟩

As it is written in the previous section, every topological group is regular.

theorem (**in** topgroup) topgroup_reg:
  shows T{is regular}
⟨*proof*⟩

The promised corollary follows:

corollary (**in** topgroup) T2_imp_T3:
  assumes T{is T_2}
  shows T{is T_3} ⟨*proof*⟩

end

93 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.

93.1 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∈ $\mathcal{N}_0$. {⟨s,t⟩∈G×G. (-s)+t ∈U} ⊆ V}

Definition of right uniformity

definition (**in** topgroup) rightUniformity
 where rightUniformity ≡ {V∈Pow(G×G).∃U∈ $\mathcal{N}_0$. {⟨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*⟩

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*⟩

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*⟩

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*⟩

The third uniformity important on topological groups is called the uniformity of Roelcke.

definition(**in** topgroup) roelckeUniformity
 where roelckeUniformity ≡ {V∈Pow(G×G). ∃U∈ $\mathcal{N}_0$. {⟨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*⟩

The topology given by the roelcke uniformity is the original topology

```
lemma (in topgroup) top_generated_roelcke_uniformity:
  shows UniformTopology(roelckeUniformity,G) = T
⟨proof⟩
```

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⟩

end
```

94 Topological groups 2

```
theory TopologicalGroup_ZF_2 imports Topology_ZF_8 TopologicalGroup_ZF
Group_ZF_2
begin
```

This theory deals with quotient topological groups.

94.1 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⟩
```

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⟩

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
  ⟨proof⟩
```

```
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⟩
```

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)
    ⟨proof⟩


end
```

95 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.

95.1 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⟩
```

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⟩
```

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⟩
```

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⟩
```

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)
  ⟨proof⟩
```

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⟩
```

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⟩
```

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⟩
```

end

96 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.

96.1 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.

96.2 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:
```

$A \in \mathbb{R} \land B \in \mathbb{R} \longrightarrow (A <_{\mathbb{R}} B \longleftrightarrow \neg(A=B \lor B <_{\mathbb{R}} A))$
assumes MMI_pre_axlttrn:
$A \in \mathbb{R} \land B \in \mathbb{R} \land C \in \mathbb{R} \longrightarrow ((A <_{\mathbb{R}} B \land B <_{\mathbb{R}} C) \longrightarrow A <_{\mathbb{R}} C)$
assumes MMI_pre_axltadd:
$A \in \mathbb{R} \land B \in \mathbb{R} \land C \in \mathbb{R} \longrightarrow (A <_{\mathbb{R}} B \longrightarrow C+A <_{\mathbb{R}} C+B)$
assumes MMI_pre_axmulgt0:
$A \in \mathbb{R} \land B \in \mathbb{R} \longrightarrow (\mathbf{0} <_{\mathbb{R}} A \land \mathbf{0} <_{\mathbb{R}} B \longrightarrow \mathbf{0} <_{\mathbb{R}} A\cdot B)$
assumes MMI_pre_axsup:
$A \subseteq \mathbb{R} \land A \neq 0 \land (\exists x\in\mathbb{R}. \forall y\in A. y <_{\mathbb{R}} x) \longrightarrow$
$(\exists x\in\mathbb{R}. (\forall y\in A. \neg(x <_{\mathbb{R}} y)) \land (\forall y\in\mathbb{R}. (y <_{\mathbb{R}} x \longrightarrow (\exists z\in A. y <_{\mathbb{R}} z))))$
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: $A \in \mathbb{C} \land B \in \mathbb{C} \longrightarrow A \cdot B = B \cdot A$
assumes MMI_axaddcl: $A \in \mathbb{C} \land B \in \mathbb{C} \longrightarrow A + B \in \mathbb{C}$
assumes MMI_axmulcl: $A \in \mathbb{C} \land B \in \mathbb{C} \longrightarrow A \cdot B \in \mathbb{C}$
assumes MMI_axdistr:
$A \in \mathbb{C} \land B \in \mathbb{C} \land C \in \mathbb{C} \longrightarrow A\cdot(B + C) = A\cdot B + A\cdot C$
assumes MMI_axaddcom: $A \in \mathbb{C} \land B \in \mathbb{C} \longrightarrow A + B = B + A$
assumes MMI_axaddass:
$A \in \mathbb{C} \land B \in \mathbb{C} \land C \in \mathbb{C} \longrightarrow A + B + C = A + (B + C)$
assumes MMI_axmulass:
$A \in \mathbb{C} \land B \in \mathbb{C} \land C \in \mathbb{C} \longrightarrow A \cdot B \cdot C = A \cdot (B \cdot C)$
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: $A \in \mathbb{C} \land A \neq \mathbf{0} \longrightarrow (\exists x \in \mathbb{C}. A \cdot x = \mathbf{1})$
assumes MMI_ax1id: $A \in \mathbb{C} \longrightarrow A \cdot \mathbf{1} = A$
assumes MMI_axaddrcl: $A \in \mathbb{R} \land B \in \mathbb{R} \longrightarrow A + B \in \mathbb{R}$
assumes MMI_axmulrcl: $A \in \mathbb{R} \land B \in \mathbb{R} \longrightarrow A \cdot B \in \mathbb{R}$
assumes MMI_axrnegex: $A \in \mathbb{R} \longrightarrow (\exists x \in \mathbb{R}. A + x = \mathbf{0})$
assumes MMI_axrrecex: $A \in \mathbb{R} \land A \neq \mathbf{0} \longrightarrow (\exists x \in \mathbb{R}. A \cdot x = \mathbf{1})$

end

97 Logic and sets in Metamatah

theory MMI_logic_and_sets **imports** MMI_prelude

begin

97.1 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** $\varphi \longrightarrow \psi$ **shows** ψ
⟨*proof*⟩

lemma `MMI_sseli`: **assumes** `A1: A` $\subseteq$ `B`
shows `C` $\in$ `A` $\longrightarrow$ `C` $\in$ `B`
⟨*proof*⟩

lemma `MMI_sselii`: **assumes** `A1: A` $\subseteq$ `B` **and**
`A2: C` $\in$ `A`
shows `C` $\in$ `B`
⟨*proof*⟩

lemma `MMI_syl`: **assumes** `A1:` φ $\longrightarrow$ `ps` **and**
`A2: ps` $\longrightarrow$ `ch`
shows φ $\longrightarrow$ `ch`
⟨*proof*⟩

lemma `MMI_elimhyp`: **assumes** `A1: A = if (` φ `, A , B )` $\longrightarrow$ `(` $\varphi \longleftrightarrow \psi$ `)` **and**
`A2: B = if (` φ `, A , B )` $\longrightarrow$ `( ch` $\longleftrightarrow \psi$ `)` **and**
`A3: ch`
shows ψ
⟨*proof*⟩

lemma `MMI_neeq1`:
shows `A = B` $\longrightarrow$ `( A` $\neq$ `C` $\longleftrightarrow$ `B` $\neq$ `C )`
⟨*proof*⟩

lemma `MMI_mp2`: **assumes** `A1:` φ **and**
`A2:` ψ **and**
`A3:` φ $\longrightarrow$ `(` ψ $\longrightarrow$ `chi )`
shows `chi`
⟨*proof*⟩

lemma `MMI_xpex`: **assumes** `A1: A isASet` **and**
`A2: B isASet`
shows `( A` $\times$ `B ) isASet`
⟨*proof*⟩

lemma `MMI_fex`:
shows

A $\in$ C $\longrightarrow$ (F : A $\rightarrow$ B $\longrightarrow$ F isASet)
A isASet $\longrightarrow$ (F : A $\rightarrow$ B $\longrightarrow$ F isASet)
⟨*proof*⟩

lemma MMI_3eqtr4d: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: φ $\longrightarrow$ C = A **and**
A3: φ $\longrightarrow$ D = B
shows φ $\longrightarrow$ C = D
⟨*proof*⟩

lemma MMI_3coml: **assumes** A1: (φ $\wedge$ ψ $\wedge$ chi) $\longrightarrow$ th
shows (ψ $\wedge$ chi $\wedge$ φ) $\longrightarrow$ th
⟨*proof*⟩

lemma MMI_sylan: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ chi **and**
A2: th $\longrightarrow$ φ
shows (th $\wedge$ ψ) $\longrightarrow$ chi
⟨*proof*⟩

lemma MMI_3impa: **assumes** A1: ((φ $\wedge$ ψ) $\wedge$ chi) $\longrightarrow$ th
shows (φ $\wedge$ ψ $\wedge$ chi) $\longrightarrow$ th
⟨*proof*⟩

lemma MMI_3adant2: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ chi
shows (φ $\wedge$ th $\wedge$ ψ) $\longrightarrow$ chi
⟨*proof*⟩

lemma MMI_3adant1: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ chi
shows (th $\wedge$ φ $\wedge$ ψ) $\longrightarrow$ chi
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_opreq12d: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: φ $\longrightarrow$ C = D
shows
φ $\longrightarrow$ (A + C) = (B + D)
φ $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ D)
φ $\longrightarrow$ (A - C) = (B - D)
φ $\longrightarrow$ (A / C) = (B / D)
⟨*proof*⟩

lemma MMI_mp2an: **assumes** A1: φ **and**
A2: ψ **and**
A3: (φ $\wedge$ ψ) $\longrightarrow$ chi
shows chi
⟨*proof*⟩

lemma MMI_mp3an: **assumes** A1: φ **and**
A2: ψ **and**
A3: ch **and**

A4: (φ ∧ ψ ∧ ch) ⟶ ϑ
shows ϑ
⟨*proof*⟩

lemma MMI_eqeltrr: **assumes** A1: A = B **and**
A2: A ∈ C
shows B ∈ C
⟨*proof*⟩

lemma MMI_eqtr: **assumes** A1: A = B **and**
A2: B = C
shows A = C
⟨*proof*⟩

lemma MMI_impbi: **assumes** A1: φ ⟶ ψ **and**
A2: ψ ⟶ φ
shows φ ⟷ ψ
⟨*proof*⟩

lemma MMI_mp3an3: **assumes** A1: ch **and**
A2: (φ ∧ ψ ∧ ch) ⟶ ϑ
shows (φ ∧ ψ) ⟶ ϑ
⟨*proof*⟩

lemma MMI_eqeq12d: **assumes** A1: φ ⟶ A = B **and**
A2: φ ⟶ C = D
shows φ ⟶ (A = C ⟷ B = D)
⟨*proof*⟩

lemma MMI_mpan2: **assumes** A1: ψ **and**
A2: (φ ∧ ψ) ⟶ ch
shows φ ⟶ ch
⟨*proof*⟩

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)
⟨*proof*⟩

lemma MMI_syl5bir: **assumes** A1: φ ⟶ (ψ ⟷ ch) **and**
A2: ϑ ⟶ ch
shows φ ⟶ (ϑ ⟶ ψ)
⟨*proof*⟩

lemma MMI_adantr: **assumes** A1: $\varphi \longrightarrow \psi$
shows ($\varphi \wedge$ ch) $\longrightarrow \psi$
$\langle proof \rangle$

lemma MMI_mpan: **assumes** A1: φ **and**
A2: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows $\psi \longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_eqeq1d: **assumes** A1: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ (A = C $\longleftrightarrow$ B = C)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_opreq1:
shows
A = B $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ C)
A = B $\longrightarrow$ (A + C) = (B + C)
A = B $\longrightarrow$ (A - C) = (B - C)
A = B $\longrightarrow$ (A / C) = (B / C)
$\langle proof \rangle$

lemma MMI_syl6eq: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: B = C
shows $\varphi \longrightarrow$ A = C
$\langle proof \rangle$

lemma MMI_syl6bi: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
A2: ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
$\langle proof \rangle$

lemma MMI_imp: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch)
shows ($\varphi \wedge \psi$) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_sylibd: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ (ch $\longleftrightarrow \vartheta$)
shows $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
$\langle proof \rangle$

lemma MMI_ex: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch)
$\langle proof \rangle$

lemma MMI_r19_23aiv: **assumes** A1: $\forall$x. (x $\in$ A $\longrightarrow$ (φ(x) $\longrightarrow \psi$))
shows ($\exists$ x $\in$ A . φ(x)) $\longrightarrow \psi$
$\langle proof \rangle$

lemma MMI_bitr: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**

A2: ψ $\longleftrightarrow$ ch
shows φ $\longleftrightarrow$ ch
⟨*proof*⟩

lemma MMI_eqeq12i: **assumes** A1: A = B **and**
A2: C = D
shows A = C $\longleftrightarrow$ B = D
⟨*proof*⟩

lemma MMI_dedth3h:
assumes A1: A = if (φ , A , D) $\longrightarrow$ (ϑ $\longleftrightarrow$ ta) **and**
A2: B = if (ψ , B , R) $\longrightarrow$ (ta $\longleftrightarrow$ et) **and**
A3: C = if (ch , C , S) $\longrightarrow$ (et $\longleftrightarrow$ ze) **and**
A4: ze
shows (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_bibi1d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ((ψ $\longleftrightarrow$ ϑ) $\longleftrightarrow$ (ch $\longleftrightarrow$ ϑ))
⟨*proof*⟩

lemma MMI_eqeq1:
shows A = B $\longrightarrow$ (A = C $\longleftrightarrow$ B = C)
⟨*proof*⟩

lemma MMI_bibi12d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: φ $\longrightarrow$ (ϑ $\longleftrightarrow$ ta)
shows φ $\longrightarrow$ ((ψ $\longleftrightarrow$ ϑ) $\longleftrightarrow$ (ch $\longleftrightarrow$ ta))
⟨*proof*⟩

lemma MMI_eqeq2d: **assumes** A1: φ $\longrightarrow$ A = B
shows φ $\longrightarrow$ (C = A $\longleftrightarrow$ C = B)
⟨*proof*⟩

lemma MMI_eqeq2:
shows A = B $\longrightarrow$ (C = A $\longleftrightarrow$ C = B)
⟨*proof*⟩

lemma MMI_elimel: **assumes** A1: B $\in$ C
shows if (A $\in$ C , A , B) $\in$ C
⟨*proof*⟩

lemma MMI_3adant3: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch
shows (φ $\wedge$ ψ $\wedge$ ϑ) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_bitr3d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ϑ)
shows φ $\longrightarrow$ (ch $\longleftrightarrow$ ϑ)

⟨*proof*⟩

lemma `MMI_3eqtr3d`: **assumes** `A1`: $\varphi \longrightarrow$ `A = B` **and**
`A2`: $\varphi \longrightarrow$ `A = C` **and**
`A3`: $\varphi \longrightarrow$ `B = D`
shows $\varphi \longrightarrow$ `C = D`
⟨*proof*⟩

lemma (in `MMIsar0`**)** `MMI_opreq1d`: **assumes** `A1`: $\varphi \longrightarrow$ `A = B`
shows
$\varphi \longrightarrow$ `( A + C ) = ( B + C )`
$\varphi \longrightarrow$ `( A - C ) = ( B - C )`
$\varphi \longrightarrow$ `( A · C ) = ( B · C )`
$\varphi \longrightarrow$ `( A / C ) = ( B / C )`
⟨*proof*⟩

lemma `MMI_3com12`: **assumes** `A1`: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows ($\psi \wedge \varphi \wedge$ ch) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma (in `MMIsar0`**)** `MMI_opreq2d`: **assumes** `A1`: $\varphi \longrightarrow$ `A = B`
shows
$\varphi \longrightarrow$ `( C + A ) = ( C + B )`
$\varphi \longrightarrow$ `( C - A ) = ( C - B )`
$\varphi \longrightarrow$ `( C · A ) = ( C · B )`
$\varphi \longrightarrow$ `( C / A ) = ( C / B )`
⟨*proof*⟩

lemma `MMI_3com23`: **assumes** `A1`: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows ($\varphi \wedge$ ch $\wedge \psi$) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma `MMI_3expa`: **assumes** `A1`: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma `MMI_adantrr`: **assumes** `A1`: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows ($\varphi \wedge$ ($\psi \wedge \vartheta$)) $\longrightarrow$ ch
⟨*proof*⟩

lemma `MMI_3expb`: **assumes** `A1`: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma `MMI_an4s`: **assumes** `A1`: (($\varphi \wedge \psi$) $\wedge$ (ch $\wedge \vartheta$)) $\longrightarrow \tau$
shows (($\varphi \wedge$ ch) $\wedge$ ($\psi \wedge \vartheta$)) $\longrightarrow \tau$
⟨*proof*⟩

lemma MMI_eqtrd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ B = C
shows $\varphi \longrightarrow$ A = C
⟨*proof*⟩

lemma MMI_ad2ant2l: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows (($\vartheta \wedge \varphi$) $\wedge$ ($\tau \wedge \psi$)) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_pm3_2i: **assumes** A1: φ **and**
A2: ψ
shows $\varphi \wedge \psi$
⟨*proof*⟩

lemma (in MMIsar0) MMI_opreq2i: **assumes** A1: A = B
shows
(C + A) = (C + B)
(C - A) = (C - B)
(C $\cdot$ A) = (C $\cdot$ B)
⟨*proof*⟩

lemma MMI_mpbir2an: **assumes** A1: $\varphi \longleftrightarrow$ ($\psi \wedge$ ch) **and**
A2: ψ **and**
A3: ch
shows φ
⟨*proof*⟩

lemma MMI_reu4: **assumes** A1: $\forall$x y. x = y $\longrightarrow$ (φ(x) $\longleftrightarrow$ ψ(y))
shows ($\exists$! x . x $\in$ A $\wedge$ φ(x)) $\longleftrightarrow$
(($\exists$ x $\in$ A . φ(x)) $\wedge$ ($\forall$ x $\in$ A . $\forall$ y $\in$ A .
((φ(x) $\wedge$ ψ(y)) $\longrightarrow$ x = y)))
⟨*proof*⟩

lemma MMI_risset:
shows A $\in$ B $\longleftrightarrow$ ($\exists$ x $\in$ B . x = A)
⟨*proof*⟩

lemma MMI_sylib: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_mp3an13: **assumes** A1: φ **and**
A2: ch **and**
A3: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows $\psi \longrightarrow \vartheta$

⟨*proof*⟩

lemma `MMI_eqcomd`: **assumes** A1: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ B = A
⟨*proof*⟩

lemma `MMI_sylan9eqr`: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\psi \longrightarrow$ B = C
shows ($\psi \wedge \varphi$) $\longrightarrow$ A = C
⟨*proof*⟩

lemma `MMI_exp32`: **assumes** A1: ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow \vartheta$))
⟨*proof*⟩

lemma `MMI_impcom`: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch)
shows ($\psi \wedge \varphi$) $\longrightarrow$ ch
⟨*proof*⟩

lemma `MMI_a1d`: **assumes** A1: $\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow$ (ch $\longrightarrow \psi$)
⟨*proof*⟩

lemma `MMI_r19_21aiv`: **assumes** A1: $\forall$x. $\varphi \longrightarrow$ (x $\in$ A $\longrightarrow \psi$(x))
shows $\varphi \longrightarrow$ ($\forall$ x $\in$ A . ψ(x))
⟨*proof*⟩

lemma `MMI_r19_22`:
shows ($\forall$ x $\in$ A . (φ(x) $\longrightarrow \psi$(x))) $\longrightarrow$
(($\exists$ x $\in$ A . φ(x)) $\longrightarrow$ ($\exists$ x $\in$ A . ψ(x)))
⟨*proof*⟩

lemma `MMI_syl6`: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch) **and**
A2: ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
⟨*proof*⟩

lemma `MMI_mpid`: **assumes** A1: $\varphi \longrightarrow$ ch **and**
A2: $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow \vartheta$))
shows $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
⟨*proof*⟩

lemma `MMI_eqtr3t`:
shows (A = C $\wedge$ B = C) $\longrightarrow$ A = B
⟨*proof*⟩

lemma `MMI_syl5bi`: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
A2: $\vartheta \longrightarrow \psi$
shows $\varphi \longrightarrow$ ($\vartheta \longrightarrow$ ch)

$\langle proof \rangle$

lemma MMI_mp3an1: **assumes** A1: φ **and**
A2: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows (ψ $\wedge$ ch) $\longrightarrow$ ϑ
$\langle proof \rangle$

lemma MMI_rgen2: **assumes** A1: $\forall$x y. (x $\in$ A $\wedge$ y $\in$ A) $\longrightarrow$ φ(x,y)
shows $\forall$ x $\in$ A . $\forall$ y $\in$ A . φ(x,y)
$\langle proof \rangle$

lemma MMI_ax_17: **shows** φ $\longrightarrow$ ($\forall$x. φ) $\langle proof \rangle$

lemma MMI_3eqtr4g: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: C = A **and**
A3: D = B
shows φ $\longrightarrow$ C = D
$\langle proof \rangle$

lemma MMI_3imtr4: **assumes** A1: φ $\longrightarrow$ ψ **and**
A2: ch $\longleftrightarrow$ φ **and**
A3: ϑ $\longleftrightarrow$ ψ
shows ch $\longrightarrow$ ϑ
$\langle proof \rangle$

lemma MMI_eleq2i: **assumes** A1: A = B
shows C $\in$ A $\longleftrightarrow$ C $\in$ B
$\langle proof \rangle$

lemma MMI_albii: **assumes** A1: φ $\longleftrightarrow$ ψ
shows ($\forall$ x . φ) $\longleftrightarrow$ ($\forall$ x . ψ)
$\langle proof \rangle$

lemma MMI_reucl:
shows ($\exists$! x . x $\in$ A $\wedge$ φ(x)) $\longrightarrow$ $\bigcup$ { x $\in$ A . φ(x) } $\in$ A
$\langle proof \rangle$

lemma MMI_dedth2h: **assumes** A1: A = if (φ , A , C) $\longrightarrow$ (ch $\longleftrightarrow$ ϑ) **and**
A2: B = if (ψ , B , D) $\longrightarrow$ (ϑ $\longleftrightarrow$ τ) **and**
A3: τ

shows (φ $\wedge$ ψ) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_eleq1d: **assumes** A1: φ $\longrightarrow$ A = B
shows φ $\longrightarrow$ (A $\in$ C $\longleftrightarrow$ B $\in$ C)
$\langle proof \rangle$

lemma MMI_syl5eqel: **assumes** A1: φ $\longrightarrow$ A $\in$ B **and**
A2: C = A
shows φ $\longrightarrow$ C $\in$ B
$\langle proof \rangle$

lemma IML_eeuni: **assumes** A1: x $\in$ A **and** A2: $\exists$! t . t $\in$ A $\wedge$ φ(t)
shows φ(x) $\longleftrightarrow$ $\bigcup$ { x $\in$ A . φ(x) } = x
$\langle proof \rangle$

lemma MMI_reuuni1:
shows (x $\in$ A $\wedge$ ($\exists$! x . x $\in$ A $\wedge$ φ(x))) $\longrightarrow$
(φ(x) $\longleftrightarrow$ $\bigcup$ { x $\in$ A . φ(x) } = x)
$\langle proof \rangle$

lemma MMI_eqeq1i: **assumes** A1: A = B
shows A = C $\longleftrightarrow$ B = C
$\langle proof \rangle$

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))
$\langle proof \rangle$

lemma MMI_syl6rbbrA: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: ϑ $\longleftrightarrow$ ch
shows φ $\longrightarrow$ (ϑ $\longleftrightarrow$ ψ)
$\langle proof \rangle$

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
$\langle proof \rangle$

lemma MMI_3bitr4: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
A2: ch $\longleftrightarrow$ φ **and**
A3: ϑ $\longleftrightarrow$ ψ

shows ch $\longleftrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_mpbii: **assumes** Amin: ψ **and**
Amaj: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_eqid:
shows A = A
⟨*proof*⟩

lemma MMI_pm3_27:
shows ($\varphi \wedge \psi$) $\longrightarrow \psi$
⟨*proof*⟩

lemma MMI_pm3_26:
shows ($\varphi \wedge \psi$) $\longrightarrow \varphi$
⟨*proof*⟩

lemma MMI_ancoms: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch
shows ($\psi \wedge \varphi$) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_syl3anc: **assumes** A1: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \varphi$ **and**
A3: $\tau \longrightarrow \psi$ **and**
A4: $\tau \longrightarrow$ ch
shows $\tau \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_syl5eq: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: C = A
shows $\varphi \longrightarrow$ C = B
⟨*proof*⟩

lemma MMI_eqcomi: **assumes** A1: A = B
shows B = A
⟨*proof*⟩

lemma MMI_3eqtr: **assumes** A1: A = B **and**
A2: B = C **and**
A3: C = D
shows A = D
⟨*proof*⟩

lemma MMI_mpbir: **assumes** Amin: ψ **and**
Amaj: $\varphi \longleftrightarrow \psi$
shows φ

⟨*proof*⟩

lemma MMI_syl3an3: **assumes** A1: (φ ∧ ψ ∧ ch) ⟶ ϑ **and**
A2: τ ⟶ ch
shows (φ ∧ ψ ∧ τ) ⟶ ϑ
⟨*proof*⟩

lemma MMI_3eqtrd: **assumes** A1: φ ⟶ A = B **and**
A2: φ ⟶ B = C **and**
A3: φ ⟶ C = D
shows φ ⟶ A = D
⟨*proof*⟩

lemma MMI_syl5: **assumes** A1: φ ⟶ (ψ ⟶ ch) **and**
A2: ϑ ⟶ ψ
shows φ ⟶ (ϑ ⟶ ch)
⟨*proof*⟩

lemma MMI_exp3a: **assumes** A1: φ ⟶ ((ψ ∧ ch) ⟶ ϑ)
shows φ ⟶ (ψ ⟶ (ch ⟶ ϑ))
⟨*proof*⟩

lemma MMI_com12: **assumes** A1: φ ⟶ (ψ ⟶ ch)
shows ψ ⟶ (φ ⟶ ch)
⟨*proof*⟩

lemma MMI_3imp: **assumes** A1: φ ⟶ (ψ ⟶ (ch ⟶ ϑ))
shows (φ ∧ ψ ∧ ch) ⟶ ϑ
⟨*proof*⟩

lemma MMI_3eqtr3: **assumes** A1: A = B **and**
A2: A = C **and**
A3: B = D
shows C = D
⟨*proof*⟩

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)
⟨*proof*⟩

lemma MMI_eqtr3: **assumes** A1: A = B **and**
A2: A = C
shows B = C

⟨*proof*⟩

lemma MMI_dedth: **assumes** A1: A = if (φ , A , B) $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: ch
shows φ $\longrightarrow$ ψ
⟨*proof*⟩

lemma MMI_id:
shows φ $\longrightarrow$ φ
⟨*proof*⟩

lemma MMI_eqtr3d: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: φ $\longrightarrow$ A = C
shows φ $\longrightarrow$ B = C
⟨*proof*⟩

lemma MMI_sylan2: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch **and**
A2: ϑ $\longrightarrow$ ψ
shows (φ $\wedge$ ϑ) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_adantl: **assumes** A1: φ $\longrightarrow$ ψ
shows (ch $\wedge$ φ) $\longrightarrow$ ψ
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_opreq12:
shows
(A = B $\wedge$ C = D) $\longrightarrow$ (A + C) = (B + D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A - C) = (B - D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A $\cdot$ C) = (B $\cdot$ D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A / C) = (B / D)
⟨*proof*⟩

lemma MMI_anidms: **assumes** A1: (φ $\wedge$ φ) $\longrightarrow$ ψ
shows φ $\longrightarrow$ ψ
⟨*proof*⟩

lemma MMI_anabsan2: **assumes** A1: (φ $\wedge$ (ψ $\wedge$ ψ)) $\longrightarrow$ ch
shows (φ $\wedge$ ψ) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_3simp2:
shows (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ψ
⟨*proof*⟩

lemma MMI_3simp3:

shows (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_sylbir: **assumes** A1: ψ $\longleftrightarrow$ φ **and**
A2: ψ $\longrightarrow$ ch
shows φ $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_3eqtr3g: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: A = C **and**
A3: B = D
shows φ $\longrightarrow$ C = D
⟨*proof*⟩

lemma MMI_3bitr: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
A2: ψ $\longleftrightarrow$ ch **and**
A3: ch $\longleftrightarrow$ ϑ
shows φ $\longleftrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_3bitr3: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
A2: φ $\longleftrightarrow$ ch **and**
A3: ψ $\longleftrightarrow$ ϑ
shows ch $\longleftrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_eqcom:
shows A = B $\longleftrightarrow$ B = A
⟨*proof*⟩

lemma MMI_syl6bb: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: ch $\longleftrightarrow$ ϑ
shows φ $\longrightarrow$ (ψ $\longleftrightarrow$ ϑ)
⟨*proof*⟩

lemma MMI_3bitr3d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ϑ) **and**
A3: φ $\longrightarrow$ (ch $\longleftrightarrow$ τ)
shows φ $\longrightarrow$ (ϑ $\longleftrightarrow$ τ)
⟨*proof*⟩

lemma MMI_syl3an2: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ **and**
A2: τ $\longrightarrow$ ψ
shows (φ $\wedge$ τ $\wedge$ ch) $\longrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_df_rex:
  shows ($\exists$ x $\in$ A . φ(x)) $\longleftrightarrow$ ($\exists$ x . (x $\in$ A $\wedge$ φ(x)))
  $\langle proof \rangle$

lemma MMI_mpbi: **assumes** Amin: φ **and**
  Amaj: $\varphi \longleftrightarrow \psi$
  shows ψ
  $\langle proof \rangle$

lemma MMI_mp3an12: **assumes** A1: φ **and**
  A2: ψ **and**
  A3: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
  shows ch $\longrightarrow \vartheta$
  $\langle proof \rangle$

lemma MMI_syl5bb: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
  A2: $\vartheta \longleftrightarrow \psi$
  shows $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow$ ch)
  $\langle proof \rangle$

lemma MMI_eleq1a:
  shows A $\in$ B $\longrightarrow$ (C = A $\longrightarrow$ C $\in$ B)
  $\langle proof \rangle$

lemma MMI_sylbird: **assumes** A1: $\varphi \longrightarrow$ (ch $\longleftrightarrow \psi$) **and**
  A2: $\varphi \longrightarrow$ (ch $\longrightarrow \vartheta$)
  shows $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
  $\langle proof \rangle$

lemma MMI_19_23aiv: **assumes** A1: $\forall$x. φ(x) $\longrightarrow \psi$
  shows ($\exists$ x . φ(x)) $\longrightarrow \psi$
  $\langle proof \rangle$

lemma MMI_eqeltrrd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
  A2: $\varphi \longrightarrow$ A $\in$ C
  shows $\varphi \longrightarrow$ B $\in$ C
  $\langle proof \rangle$

lemma MMI_syl2an: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch **and**
  A2: $\vartheta \longrightarrow \varphi$ **and**
  A3: $\tau \longrightarrow \psi$
  shows ($\vartheta \wedge \tau$) $\longrightarrow$ ch
  $\langle proof \rangle$

lemma MMI_adantrl: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch
  shows ($\varphi \wedge$ ($\vartheta \wedge \psi$)) $\longrightarrow$ ch

$\langle proof \rangle$

lemma MMI_ad2ant2r: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch
shows ((φ $\wedge$ ϑ) $\wedge$ (ψ $\wedge$ τ)) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_adantll: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch
shows ((ϑ $\wedge$ φ) $\wedge$ ψ) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_anandirs: **assumes** A1: ((φ $\wedge$ ch) $\wedge$ (ψ $\wedge$ ch)) $\longrightarrow$ τ
shows ((φ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ τ
$\langle proof \rangle$

lemma MMI_adantlr: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch
shows ((φ $\wedge$ ϑ) $\wedge$ ψ) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_an42s: **assumes** A1: ((φ $\wedge$ ψ) $\wedge$ (ch $\wedge$ ϑ)) $\longrightarrow$ τ
shows ((φ $\wedge$ ch) $\wedge$ (ϑ $\wedge$ ψ)) $\longrightarrow$ τ
$\langle proof \rangle$

lemma MMI_mp3an2: **assumes** A1: ψ **and**
A2: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows (φ $\wedge$ ch) $\longrightarrow$ ϑ
$\langle proof \rangle$

lemma MMI_3simp1:
shows (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ φ
$\langle proof \rangle$

lemma MMI_3impb: **assumes** A1: (φ $\wedge$ (ψ $\wedge$ ch)) $\longrightarrow$ ϑ
shows (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
$\langle proof \rangle$

lemma MMI_mpbird: **assumes** Amin: φ $\longrightarrow$ ch **and**
Amaj: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ψ
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_opreq12i: **assumes** A1: A = B **and**
A2: C = D
shows
(A + C) = (B + D)
(A $\cdot$ C) = (B $\cdot$ D)

(A - C) = (B - D)
⟨*proof*⟩

lemma MMI_3eqtr4: **assumes** A1: A = B **and**
A2: C = A **and**
A3: D = B
shows C = D
⟨*proof*⟩

lemma MMI_eqtr4d: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ C = B
shows $\varphi \longrightarrow$ A = C
⟨*proof*⟩

lemma MMI_3eqtr3rd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ A = C **and**
A3: $\varphi \longrightarrow$ B = D
shows $\varphi \longrightarrow$ D = C
⟨*proof*⟩

lemma MMI_sylanc: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch **and**
A2: $\vartheta \longrightarrow \varphi$ **and**
A3: $\vartheta \longrightarrow \psi$
shows $\vartheta \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_anim12i: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch $\longrightarrow \vartheta$
shows ($\varphi \wedge$ ch) $\longrightarrow$ ($\psi \wedge \vartheta$)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma MMI_sylanr2: **assumes** A1: ($\varphi \wedge$ ($\psi \wedge$ ch)) $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow$ ch
shows ($\varphi \wedge$ ($\psi \wedge \tau$)) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_sylanl2: **assumes** A1: $((\varphi \wedge \psi) \wedge ch) \longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \psi$
shows $((\varphi \wedge \tau) \wedge ch) \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_ancom2s: **assumes** A1: $(\varphi \wedge (\psi \wedge ch)) \longrightarrow \vartheta$
shows $(\varphi \wedge (ch \wedge \psi)) \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_anandis: **assumes** A1: $((\varphi \wedge \psi) \wedge (\varphi \wedge ch)) \longrightarrow \tau$
shows $(\varphi \wedge (\psi \wedge ch)) \longrightarrow \tau$
⟨*proof*⟩

lemma MMI_sylan9eq: **assumes** A1: $\varphi \longrightarrow A = B$ **and**
A2: $\psi \longrightarrow B = C$
shows $(\varphi \wedge \psi) \longrightarrow A = C$
⟨*proof*⟩

lemma MMI_keephyp: **assumes** A1: $A = if (\varphi , A , B) \longrightarrow (\psi \longleftrightarrow \vartheta)$
and
A2: $B = if (\varphi , A , B) \longrightarrow (ch \longleftrightarrow \vartheta)$ **and**
A3: ψ **and**
A4: ch
shows ϑ
⟨*proof*⟩

lemma MMI_eleq1:
shows $A = B \longrightarrow (A \in C \longleftrightarrow B \in C)$
⟨*proof*⟩

lemma MMI_pm4_2i:
shows $\varphi \longrightarrow (\psi \longleftrightarrow \psi)$
⟨*proof*⟩

lemma MMI_3anbi123d: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow ch)$ **and**
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$ **and**
A3: $\varphi \longrightarrow (\eta \longleftrightarrow \zeta)$
shows $\varphi \longrightarrow ((\psi \wedge \vartheta \wedge \eta) \longleftrightarrow (ch \wedge \tau \wedge \zeta))$
⟨*proof*⟩

lemma MMI_imbi12d: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow ch)$ **and**
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
shows $\varphi \longrightarrow ((\psi \longrightarrow \vartheta) \longleftrightarrow (ch \longrightarrow \tau))$
⟨*proof*⟩

lemma MMI_bitrd: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow ch)$ **and**

A2: $\varphi \longrightarrow$ (ch $\longleftrightarrow \vartheta$)
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow \vartheta$)
⟨*proof*⟩

lemma MMI_df_ne:
shows (A $\neq$ B $\longleftrightarrow$ $\neg$ (A = B))
⟨*proof*⟩

lemma MMI_3pm3_2i: **assumes** A1: φ **and**
A2: ψ **and**
A3: ch
shows $\varphi \wedge \psi \wedge$ ch
⟨*proof*⟩

lemma MMI_eqeq2i: **assumes** A1: A = B
shows C = A $\longleftrightarrow$ C = B
⟨*proof*⟩

lemma MMI_syl5bbr: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
A2: $\psi \longleftrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow$ ch)
⟨*proof*⟩

lemma MMI_biimpd: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch)
⟨*proof*⟩

lemma MMI_orrd: **assumes** A1: $\varphi \longrightarrow$ ($\neg$ (ψ) $\longrightarrow$ ch)
shows $\varphi \longrightarrow$ ($\psi \vee$ ch)
⟨*proof*⟩

lemma MMI_jaoi: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch $\longrightarrow \psi$
shows ($\varphi \vee$ ch) $\longrightarrow \psi$
⟨*proof*⟩

lemma MMI_oridm:
shows ($\varphi \vee \varphi$) $\longleftrightarrow \varphi$
⟨*proof*⟩

lemma MMI_orbi1d: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ (($\psi \vee \vartheta$) $\longleftrightarrow$ (ch $\vee \vartheta$))
⟨*proof*⟩

lemma MMI_orbi2d: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ (($\vartheta \vee \psi$) $\longleftrightarrow$ ($\vartheta \vee$ ch))
⟨*proof*⟩

lemma MMI_3bitr4g: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ **and**
A2: $\vartheta \longleftrightarrow \psi$ **and**
A3: $\tau \longleftrightarrow \mathrm{ch}$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
⟨*proof*⟩

lemma MMI_negbid: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow (\neg (\psi) \longleftrightarrow \neg (\mathrm{ch}))$
⟨*proof*⟩

lemma MMI_ioran:
shows $\neg ((\varphi \vee \psi)) \longleftrightarrow$
$(\neg (\varphi) \wedge \neg (\psi))$
⟨*proof*⟩

lemma MMI_syl6rbb: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$ **and**
A2: $\mathrm{ch} \longleftrightarrow \vartheta$
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \psi)$
⟨*proof*⟩

lemma MMI_anbi12i: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**
A2: $\mathrm{ch} \longleftrightarrow \vartheta$
shows $(\varphi \wedge \mathrm{ch}) \longleftrightarrow (\psi \wedge \vartheta)$
⟨*proof*⟩

lemma MMI_keepel: **assumes** A1: $A \in C$ **and**
A2: $B \in C$
shows if $(\varphi , A , B) \in C$
⟨*proof*⟩

lemma MMI_imbi2d: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow \mathrm{ch})$
shows $\varphi \longrightarrow ((\vartheta \longrightarrow \psi) \longleftrightarrow (\vartheta \longrightarrow \mathrm{ch}))$
⟨*proof*⟩

lemma MMI_eqeltr: **assumes** $A = B$ **and** $B \in C$
shows $A \in C$ ⟨*proof*⟩

lemma MMI_3impia: **assumes** A1: $(\varphi \wedge \psi) \longrightarrow (\mathrm{ch} \longrightarrow \vartheta)$
shows $(\varphi \wedge \psi \wedge \mathrm{ch}) \longrightarrow \vartheta$
⟨*proof*⟩

lemma `MMI_eqneqd`: **assumes** `A1`: $\varphi \longrightarrow$ (A = B $\longleftrightarrow$ C = D)
shows $\varphi \longrightarrow$ (A $\neq$ B $\longleftrightarrow$ C $\neq$ D)
⟨*proof*⟩

lemma `MMI_3ad2ant2`: **assumes** `A1`: $\varphi \longrightarrow$ ch
shows ($\psi \wedge \varphi \wedge \vartheta$) $\longrightarrow$ ch
⟨*proof*⟩

lemma `MMI_mp3anl3`: **assumes** `A1`: ch **and**
`A2`: (($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta$) $\longrightarrow \tau$
shows (($\varphi \wedge \psi$) $\wedge \vartheta$) $\longrightarrow \tau$
⟨*proof*⟩

lemma `MMI_bitr4d`: **assumes** `A1`: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
`A2`: $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow \vartheta$)
⟨*proof*⟩

lemma `MMI_neeq1d`: **assumes** `A1`: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ (A $\neq$ C $\longleftrightarrow$ B $\neq$ C)
⟨*proof*⟩

lemma `MMI_3anim123i`: **assumes** `A1`: $\varphi \longrightarrow \psi$ **and**
`A2`: ch $\longrightarrow \vartheta$ **and**
`A3`: $\tau \longrightarrow \eta$
shows ($\varphi \wedge$ ch $\wedge \tau$) $\longrightarrow$ ($\psi \wedge \vartheta \wedge \eta$)
⟨*proof*⟩

lemma `MMI_3exp`: **assumes** `A1`: ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow \vartheta$))
⟨*proof*⟩

lemma `MMI_exp4a`: **assumes** `A1`: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ((ch $\wedge \vartheta$) $\longrightarrow \tau$))
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow$ ($\vartheta \longrightarrow \tau$)))
⟨*proof*⟩

lemma `MMI_3imp1`: **assumes** `A1`: $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow$ ($\vartheta \longrightarrow \tau$)))

shows (($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta$) $\longrightarrow \tau$
⟨*proof*⟩

lemma `MMI_anim1i`: **assumes** `A1`: $\varphi \longrightarrow \psi$
shows ($\varphi \wedge$ ch) $\longrightarrow$ ($\psi \wedge$ ch)
⟨*proof*⟩

lemma MMI_3adantl1: **assumes** A1: ((φ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ ϑ
shows ((τ $\wedge$ φ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_3adantl2: **assumes** A1: ((φ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ ϑ
shows ((φ $\wedge$ τ $\wedge$ ψ) $\wedge$ ch) $\longrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_3comr: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
shows (ch $\wedge$ φ $\wedge$ ψ) $\longrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_bitr3: **assumes** A1: ψ $\longleftrightarrow$ φ **and**
A2: ψ $\longleftrightarrow$ ch
shows φ $\longleftrightarrow$ ch
⟨*proof*⟩

lemma MMI_anbi12d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: φ $\longrightarrow$ (ϑ $\longleftrightarrow$ τ)
shows φ $\longrightarrow$ ((ψ $\wedge$ ϑ) $\longleftrightarrow$ (ch $\wedge$ τ))
⟨*proof*⟩

lemma MMI_pm3_26i: **assumes** A1: φ $\wedge$ ψ
shows φ
⟨*proof*⟩

lemma MMI_pm3_27i: **assumes** A1: φ $\wedge$ ψ
shows ψ
⟨*proof*⟩

lemma MMI_anabsan: **assumes** A1: ((φ $\wedge$ φ) $\wedge$ ψ) $\longrightarrow$ ch
shows (φ $\wedge$ ψ) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_3eqtr4rd: **assumes** A1: φ $\longrightarrow$ A = B **and**
A2: φ $\longrightarrow$ C = A **and**
A3: φ $\longrightarrow$ D = B
shows φ $\longrightarrow$ D = C
⟨*proof*⟩

lemma MMI_syl3an1: **assumes** A1: (φ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ **and**
A2: τ $\longrightarrow$ φ
shows (τ $\wedge$ ψ $\wedge$ ch) $\longrightarrow$ ϑ
⟨*proof*⟩

lemma MMI_syl3anl2: **assumes** A1: ((φ $\wedge$ ψ $\wedge$ ch) $\wedge$ ϑ) $\longrightarrow$ τ **and**
A2: η $\longrightarrow$ ψ
shows ((φ $\wedge$ η $\wedge$ ch) $\wedge$ ϑ) $\longrightarrow$ τ
$\langle proof \rangle$

lemma MMI_jca: **assumes** A1: φ $\longrightarrow$ ψ **and**
A2: φ $\longrightarrow$ ch
shows φ $\longrightarrow$ (ψ $\wedge$ ch)
$\langle proof \rangle$

lemma MMI_3ad2ant3: **assumes** A1: φ $\longrightarrow$ ch
shows (ψ $\wedge$ ϑ $\wedge$ φ) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_anim2i: **assumes** A1: φ $\longrightarrow$ ψ
shows (ch $\wedge$ φ) $\longrightarrow$ (ch $\wedge$ ψ)
$\langle proof \rangle$

lemma MMI_ancom:
shows (φ $\wedge$ ψ) $\longleftrightarrow$ (ψ $\wedge$ φ)
$\langle proof \rangle$

lemma MMI_anbi1i: **assumes** Aaa: φ $\longleftrightarrow$ ψ
shows (φ $\wedge$ ch) $\longleftrightarrow$ (ψ $\wedge$ ch)
$\langle proof \rangle$

lemma MMI_an42:
shows ((φ $\wedge$ ψ) $\wedge$ (ch $\wedge$ ϑ)) $\longleftrightarrow$
((φ $\wedge$ ch) $\wedge$ (ϑ $\wedge$ ψ))
$\langle proof \rangle$

lemma MMI_sylanb: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch **and**
A2: ϑ $\longleftrightarrow$ φ
shows (ϑ $\wedge$ ψ) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_an4:
shows ((φ $\wedge$ ψ) $\wedge$ (ch $\wedge$ ϑ)) $\longleftrightarrow$
((φ $\wedge$ ch) $\wedge$ (ψ $\wedge$ ϑ))
$\langle proof \rangle$

lemma MMI_syl2anb: **assumes** A1: (φ $\wedge$ ψ) $\longrightarrow$ ch **and**
A2: ϑ $\longleftrightarrow$ φ **and**
A3: τ $\longleftrightarrow$ ψ
shows (ϑ $\wedge$ τ) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_eqtr2d: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ B = C
shows $\varphi \longrightarrow$ C = A
⟨*proof*⟩

lemma MMI_sylbid: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ (ch $\longrightarrow \vartheta$)
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
⟨*proof*⟩

lemma MMI_sylanl1: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \varphi$
shows (($\tau \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_sylan2b: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch **and**
A2: $\vartheta \longleftrightarrow \psi$
shows ($\varphi \wedge \vartheta$) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_pm3_22:
shows ($\varphi \wedge \psi$) $\longrightarrow$ ($\psi \wedge \varphi$)
⟨*proof*⟩

lemma MMI_ancli: **assumes** A1: $\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow (\varphi \wedge \psi)$
⟨*proof*⟩

lemma MMI_ad2antlr: **assumes** A1: $\varphi \longrightarrow \psi$
shows ((ch $\wedge \varphi$) $\wedge \vartheta$) $\longrightarrow \psi$
⟨*proof*⟩

lemma MMI_biimpa: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows ($\varphi \wedge \psi$) $\longrightarrow$ ch
⟨*proof*⟩

lemma MMI_sylan2i: **assumes** A1: $\varphi \longrightarrow$ (($\psi \wedge$ ch) $\longrightarrow \vartheta$) **and**
A2: $\tau \longrightarrow$ ch
shows $\varphi \longrightarrow ((\psi \wedge \tau) \longrightarrow \vartheta)$
⟨*proof*⟩

lemma MMI_3jca: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow$ ch **and**
A3: $\varphi \longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ($\psi \wedge$ ch $\wedge \vartheta$)
⟨*proof*⟩

lemma MMI_com34: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow$ ($\vartheta \longrightarrow \tau$)))

shows $\varphi \longrightarrow (\psi \longrightarrow (\vartheta \longrightarrow (\mathtt{ch} \longrightarrow \tau)))$
⟨*proof*⟩

lemma `MMI_imp43`: **assumes** `A1`: $\varphi \longrightarrow (\psi \longrightarrow (\mathtt{ch} \longrightarrow (\vartheta \longrightarrow \tau)))$

shows $((\varphi \wedge \psi) \wedge (\mathtt{ch} \wedge \vartheta)) \longrightarrow \tau$
⟨*proof*⟩

lemma `MMI_3anass`:
shows $(\varphi \wedge \psi \wedge \mathtt{ch}) \longleftrightarrow (\varphi \wedge (\psi \wedge \mathtt{ch}))$
⟨*proof*⟩

lemma `MMI_3eqtr4r`: **assumes** `A1: A = B` **and**
`A2: C = A` **and**
`A3: D = B`
shows `D = C`
⟨*proof*⟩

lemma `MMI_jctl`: **assumes** `A1`: ψ
shows $\varphi \longrightarrow (\psi \wedge \varphi)$
⟨*proof*⟩

lemma `MMI_sylibr`: **assumes** `A1`: $\varphi \longrightarrow \psi$ **and**
`A2`: $\mathtt{ch} \longleftrightarrow \psi$
shows $\varphi \longrightarrow \mathtt{ch}$
⟨*proof*⟩

lemma `MMI_mpanl1`: **assumes** `A1`: φ **and**
`A2`: $((\varphi \wedge \psi) \wedge \mathtt{ch}) \longrightarrow \vartheta$
shows $(\psi \wedge \mathtt{ch}) \longrightarrow \vartheta$
⟨*proof*⟩

lemma `MMI_a1i`: **assumes** `A1`: φ
shows $\psi \longrightarrow \varphi$
⟨*proof*⟩

lemma **(in** `MMIsar0`**)** `MMI_opreqan12rd`: **assumes** `A1`: $\varphi \longrightarrow$ `A = B` **and**
`A2`: $\psi \longrightarrow$ `C = D`
shows
$(\psi \wedge \varphi) \longrightarrow (A + C) = (B + D)$
$(\psi \wedge \varphi) \longrightarrow (A \cdot C) = (B \cdot D)$
$(\psi \wedge \varphi) \longrightarrow (A - C) = (B - D)$
$(\psi \wedge \varphi) \longrightarrow (A / C) = (B / D)$
⟨*proof*⟩

lemma `MMI_3adantl3`: **assumes** `A1`: $((\varphi \wedge \psi) \wedge \mathtt{ch}) \longrightarrow \vartheta$

shows ((φ ∧ ψ ∧ τ) ∧ ch) ⟶ ϑ
⟨*proof*⟩

lemma MMI_sylbi: **assumes** A1: φ ⟷ ψ **and**
A2: ψ ⟶ ch
shows φ ⟶ ch
⟨*proof*⟩

lemma MMI_eirr:
shows ¬ (A ∈ A)
⟨*proof*⟩

lemma MMI_eleq1i: **assumes** A1: A = B
shows A ∈ C ⟷ B ∈ C
⟨*proof*⟩

lemma MMI_mtbir: **assumes** A1: ¬ (ψ) **and**
A2: φ ⟷ ψ
shows ¬ (φ)
⟨*proof*⟩

lemma MMI_mto: **assumes** A1: ¬ (ψ) **and**
A2: φ ⟶ ψ
shows ¬ (φ)
⟨*proof*⟩

lemma MMI_df_nel:
shows (A ∉ B ⟷ ¬ (A ∈ B))
⟨*proof*⟩

lemma MMI_snid: **assumes** A1: A isASet
shows A ∈ { A }
⟨*proof*⟩

lemma MMI_en2lp:
shows ¬ (A ∈ B ∧ B ∈ A)
⟨*proof*⟩

lemma MMI_imnan:
shows (φ ⟶ ¬ (ψ)) ⟷ ¬ ((φ ∧ ψ))
⟨*proof*⟩

lemma MMI_sseqtr4: **assumes** A1: A ⊆ B **and**
A2: C = B
shows A ⊆ C

⟨*proof*⟩

lemma MMI_ssun1:
shows A $\subseteq$ (A $\cup$ B)
⟨*proof*⟩

lemma MMI_ibar:
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ($\varphi \wedge \psi$))
⟨*proof*⟩

lemma MMI_mtbiri: **assumes** Amin: $\neg$ (ch) **and**
Amaj: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow \neg$ (ψ)
⟨*proof*⟩

lemma MMI_con2i: **assumes** Aa: $\varphi \longrightarrow \neg$ (ψ)
shows $\psi \longrightarrow \neg$ (φ)
⟨*proof*⟩

lemma MMI_intnand: **assumes** A1: $\varphi \longrightarrow \neg$ (ψ)
shows $\varphi \longrightarrow \neg$ ((ch $\wedge$ ψ))
⟨*proof*⟩

lemma MMI_intnanrd: **assumes** A1: $\varphi \longrightarrow \neg$ (ψ)
shows $\varphi \longrightarrow \neg$ ((ψ $\wedge$ ch))
⟨*proof*⟩

lemma MMI_biorf:
shows $\neg$ (φ) $\longrightarrow$ ($\psi \longleftrightarrow$ ($\varphi \vee \psi$))
⟨*proof*⟩

lemma MMI_bitr2d: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ (ch $\longleftrightarrow \vartheta$)
shows $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow \psi$)
⟨*proof*⟩

lemma MMI_orass:
shows (($\varphi \vee \psi$) $\vee$ ch) $\longleftrightarrow$ ($\varphi \vee$ (ψ $\vee$ ch))
⟨*proof*⟩

lemma MMI_orcom:
shows ($\varphi \vee \psi$) $\longleftrightarrow$ ($\psi \vee \varphi$)
⟨*proof*⟩

lemma MMI_3bitr4d: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ ($\vartheta \longleftrightarrow \psi$) **and**

A3: $\varphi \longrightarrow (\tau \longleftrightarrow$ ch)
shows $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
⟨*proof*⟩

lemma MMI_3imtr4d: **assumes** A1: $\varphi \longrightarrow (\psi \longrightarrow$ ch) **and**
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \psi)$ **and**
A3: $\varphi \longrightarrow (\tau \longleftrightarrow$ ch)
shows $\varphi \longrightarrow (\vartheta \longrightarrow \tau)$
⟨*proof*⟩

lemma MMI_3impdi: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ($\varphi \wedge$ ch)) $\longrightarrow \vartheta$
shows ($\varphi \wedge \psi \wedge$ ch) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_bi2anan9: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch) **and**
A2: $\vartheta \longrightarrow (\tau \longleftrightarrow \eta)$
shows ($\varphi \wedge \vartheta$) $\longrightarrow$ (($\psi \wedge \tau$) $\longleftrightarrow$ (ch $\wedge \eta$))
⟨*proof*⟩

lemma MMI_ssel2:
shows ((A $\subseteq$ B $\wedge$ C $\in$ A) $\longrightarrow$ C $\in$ B)
⟨*proof*⟩

lemma MMI_an1rs: **assumes** A1: (($\varphi \wedge \psi$) $\wedge$ ch) $\longrightarrow \vartheta$
shows (($\varphi \wedge$ ch) $\wedge \psi$) $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_ralbidva: **assumes** A1: $\forall$x. ($\varphi \wedge$ x $\in$ A) $\longrightarrow$ (ψ(x) $\longleftrightarrow$ ch(x))
shows $\varphi \longrightarrow$ (($\forall$ x $\in$ A . ψ(x)) $\longleftrightarrow$ ($\forall$ x $\in$ A . ch(x)))
⟨*proof*⟩

lemma MMI_rexbidva: **assumes** A1: $\forall$x. ($\varphi \wedge$ x $\in$ A) $\longrightarrow$ (ψ(x) $\longleftrightarrow$ ch(x))
shows $\varphi \longrightarrow$ (($\exists$ x $\in$ A . ψ(x)) $\longleftrightarrow$ ($\exists$ x $\in$ A . ch(x)))
⟨*proof*⟩

lemma MMI_con2bid: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow \neg$ (ch))
shows $\varphi \longrightarrow$ (ch $\longleftrightarrow \neg (\psi)$)
⟨*proof*⟩

lemma MMI_so: **assumes**
A1: $\forall$x y z. (x $\in$ A $\wedge$ y $\in$ A $\wedge$ z $\in$ A) $\longrightarrow$
((⟨x,y⟩ $\in$ R $\longleftrightarrow \neg$ ((x = y $\vee$ ⟨y, x⟩ $\in$ R))) $\wedge$

(($\langle$x, y$\rangle$ $\in$ R $\wedge$ $\langle$y, z$\rangle$ $\in$ R) $\longrightarrow$ $\langle$x, z$\rangle$ $\in$ R))
shows R Orders A
$\langle$*proof*$\rangle$

lemma MMI_con1bid: **assumes** A1: φ $\longrightarrow$ ($\neg$ (ψ) $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ($\neg$ (ch) $\longleftrightarrow$ ψ)
$\langle$*proof*$\rangle$

lemma MMI_sotrieq:
shows ((R Orders A) $\wedge$ (B $\in$ A $\wedge$ C $\in$ A)) $\longrightarrow$
(B = C $\longleftrightarrow$ $\neg$ (($\langle$B,C$\rangle$ $\in$ R $\vee$ $\langle$C, B$\rangle$ $\in$ R)))
$\langle$*proof*$\rangle$

lemma MMI_bicomd: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ (ch $\longleftrightarrow$ ψ)
$\langle$*proof*$\rangle$

lemma MMI_sotrieq2:
shows (R Orders A $\wedge$ (B $\in$ A $\wedge$ C $\in$ A)) $\longrightarrow$
(B = C $\longleftrightarrow$ ($\neg$ ($\langle$B, C$\rangle$ $\in$ R) $\wedge$ $\neg$ ($\langle$C, B$\rangle$ $\in$ R)))
$\langle$*proof*$\rangle$

lemma MMI_orc:
shows φ $\longrightarrow$ (φ $\vee$ ψ)
$\langle$*proof*$\rangle$

lemma MMI_syl6bbr: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: ϑ $\longleftrightarrow$ ch
shows φ $\longrightarrow$ (ψ $\longleftrightarrow$ ϑ)
$\langle$*proof*$\rangle$

lemma MMI_orbi1i: **assumes** A1: φ $\longleftrightarrow$ ψ
shows (φ $\vee$ ch) $\longleftrightarrow$ (ψ $\vee$ ch)
$\langle$*proof*$\rangle$

lemma MMI_syl5rbbr: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch) **and**
A2: ψ $\longleftrightarrow$ ϑ
shows φ $\longrightarrow$ (ch $\longleftrightarrow$ ϑ)
$\langle$*proof*$\rangle$

lemma MMI_anbi2d: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
shows φ $\longrightarrow$ ((ϑ $\wedge$ ψ) $\longleftrightarrow$ (ϑ $\wedge$ ch))
$\langle$*proof*$\rangle$

lemma MMI_ord: **assumes** A1: φ $\longrightarrow$ (ψ $\vee$ ch)

shows $\varphi \longrightarrow$ ($\neg$ (ψ) $\longrightarrow$ ch)
⟨*proof*⟩

lemma MMI_impbid: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ (ch $\longrightarrow \psi$)
shows $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
⟨*proof*⟩

lemma MMI_jcad: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ ($\psi \longrightarrow \vartheta$)
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\wedge \vartheta$))
⟨*proof*⟩

lemma MMI_ax_1:
shows $\varphi \longrightarrow$ ($\psi \longrightarrow \varphi$)
⟨*proof*⟩

lemma MMI_pm2_24:
shows $\varphi \longrightarrow$ ($\neg$ (φ) $\longrightarrow \psi$)
⟨*proof*⟩

lemma MMI_imp3a: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow \vartheta$))
shows $\varphi \longrightarrow$ (($\psi \wedge$ ch) $\longrightarrow \vartheta$)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breq1:
shows
A = B $\longrightarrow$ (A $\leq$ C $\longleftrightarrow$ B $\leq$ C)
A = B $\longrightarrow$ (A < C $\longleftrightarrow$ B < C)
⟨*proof*⟩

lemma MMI_biimprd: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow$ (ch $\longrightarrow \psi$)
⟨*proof*⟩

lemma MMI_jaod: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ ($\vartheta \longrightarrow$ ch)
shows $\varphi \longrightarrow$ (($\psi \vee \vartheta$) $\longrightarrow$ ch)
⟨*proof*⟩

lemma MMI_com23: **assumes** A1: $\varphi \longrightarrow$ ($\psi \longrightarrow$ (ch $\longrightarrow \vartheta$))
shows $\varphi \longrightarrow$ (ch $\longrightarrow$ ($\psi \longrightarrow \vartheta$))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breq2:
shows
A = B $\longrightarrow$ (C $\leq$ A $\longleftrightarrow$ C $\leq$ B)
A = B $\longrightarrow$ (C < A $\longleftrightarrow$ C < B)
⟨*proof*⟩

lemma MMI_syld: **assumes** A1: $\varphi \longrightarrow (\psi \longrightarrow$ ch) **and**
A2: $\varphi \longrightarrow$ (ch $\longrightarrow \vartheta$)
shows $\varphi \longrightarrow (\psi \longrightarrow \vartheta)$
⟨*proof*⟩

lemma MMI_biimpcd: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\psi \longrightarrow (\varphi \longrightarrow$ ch)
⟨*proof*⟩

lemma MMI_mp2and: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow$ ch **and**
A3: $\varphi \longrightarrow$ (($\psi \wedge$ ch) $\longrightarrow \vartheta$)
shows $\varphi \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_sonr:
shows (R Orders A $\wedge$ B $\in$ A) $\longrightarrow \neg$ (⟨B,B⟩ $\in$ R)
⟨*proof*⟩

lemma MMI_orri: **assumes** A1: $\neg (\varphi) \longrightarrow \psi$
shows $\varphi \vee \psi$
⟨*proof*⟩

lemma MMI_mpbiri: **assumes** Amin: ch **and**
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow \psi$
⟨*proof*⟩

lemma MMI_pm2_46:
shows $\neg ((\varphi \vee \psi)) \longrightarrow \neg (\psi)$
⟨*proof*⟩

lemma MMI_elun:
shows A $\in$ (B $\cup$ C) $\longleftrightarrow$ (A $\in$ B $\vee$ A $\in$ C)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_pnfxr:
shows $+\infty \in \mathbb{R}^*$
⟨*proof*⟩

lemma MMI_elisseti: **assumes** A1: A $\in$ B
shows A isASet
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mnfxr:
shows $-\infty \in \mathbb{R}^*$

⟨*proof*⟩

lemma MMI_elpr2: **assumes** A1: B isASet **and**
A2: C isASet
shows A ∈ { B , C } ⟷ (A = B ∨ A = C)
⟨*proof*⟩

lemma MMI_orbi2i: **assumes** A1: φ ⟷ ψ
shows (ch ∨ φ) ⟷ (ch ∨ ψ)
⟨*proof*⟩

lemma MMI_3orass:
shows (φ ∨ ψ ∨ ch) ⟷ (φ ∨ (ψ ∨ ch))
⟨*proof*⟩

lemma MMI_bitr4: **assumes** A1: φ ⟷ ψ **and**
A2: ch ⟷ ψ
shows φ ⟷ ch
⟨*proof*⟩

lemma MMI_eleq2:
shows A = B ⟶ (C ∈ A ⟷ C ∈ B)
⟨*proof*⟩

lemma MMI_nelneq:
shows (A ∈ C ∧ ¬ (B ∈ C)) ⟶ ¬ (A = B)
⟨*proof*⟩

lemma MMI_df_pr:
shows { A , B } = ({ A } ∪ { B })
⟨*proof*⟩

lemma MMI_ineq2i: **assumes** A1: A = B
shows (C ∩ A) = (C ∩ B)
⟨*proof*⟩

lemma MMI_mt2: **assumes** A1: ψ **and**
A2: φ ⟶ ¬ (ψ)
shows ¬ (φ)
⟨*proof*⟩

lemma MMI_disjsn:
shows (A ∩ { B }) = 0 ⟷ ¬ (B ∈ A)
⟨*proof*⟩

lemma MMI_undisj2:
shows ((A ∩ B) =

0 $\wedge$ (A $\cap$ C) =
0) $\longleftrightarrow$ (A $\cap$ (B $\cup$ C)) = 0
⟨*proof*⟩

lemma MMI_disjssun:
shows ((A $\cap$ B) = 0 $\longrightarrow$ (A $\subseteq$ (B $\cup$ C) $\longleftrightarrow$ A $\subseteq$ C))
⟨*proof*⟩

lemma MMI_uncom:
shows (A $\cup$ B) = (B $\cup$ A)
⟨*proof*⟩

lemma MMI_sseq2i: **assumes** A1: A = B
shows (C $\subseteq$ A $\longleftrightarrow$ C $\subseteq$ B)
⟨*proof*⟩

lemma MMI_disj:
shows (A $\cap$ B) =
0 $\longleftrightarrow$ ($\forall$ x $\in$ A . $\neg$ (x $\in$ B))
⟨*proof*⟩

lemma MMI_syl5ibr: **assumes** A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch) **and**
A2: ψ $\longleftrightarrow$ ϑ
shows φ $\longrightarrow$ (ϑ $\longrightarrow$ ch)
⟨*proof*⟩

lemma MMI_con3d: **assumes** A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch)
shows φ $\longrightarrow$ ($\neg$ (ch) $\longrightarrow$ $\neg$ (ψ))
⟨*proof*⟩

lemma MMI_dfrex2:
shows ($\exists$ x $\in$ A . φ(x)) $\longleftrightarrow$ $\neg$ (($\forall$ x $\in$ A . $\neg$ φ(x)))
⟨*proof*⟩

lemma MMI_visset:
shows x isASet
⟨*proof*⟩

lemma MMI_elpr: **assumes** A1: A isASet
shows A $\in$ { B , C } $\longleftrightarrow$ (A = B $\vee$ A = C)
⟨*proof*⟩

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))
⟨*proof*⟩

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))))
⟨*proof*⟩

lemma MMI_exancom:
shows ($\exists$ x . (φ(x) $\wedge$ ψ(x))) $\longleftrightarrow$
($\exists$ x . (ψ(x) $\wedge$ φ(x)))
⟨*proof*⟩

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)
⟨*proof*⟩

lemma MMI_orbi12i_orig: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
A2: ch $\longleftrightarrow$ ϑ
shows (φ $\vee$ ch) $\longleftrightarrow$ (ψ $\vee$ ϑ)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma MMI_syl6ib: **assumes** A1: φ $\longrightarrow$ (ψ $\longrightarrow$ ch) **and**
A2: ch $\longleftrightarrow$ ϑ
shows φ $\longrightarrow$ (ψ $\longrightarrow$ ϑ)
⟨*proof*⟩

lemma MMI_intnan: **assumes** A1: $\neg$ (φ)
shows $\neg$ ((ψ $\wedge$ φ))
⟨*proof*⟩

lemma MMI_intnanr: **assumes** A1: $\neg$ (φ)
shows $\neg$ ((φ $\wedge$ ψ))
⟨*proof*⟩

lemma MMI_pm3_2ni: **assumes** A1: $\neg$ (φ) **and**
A2: $\neg$ (ψ)
shows $\neg$ ((φ $\vee$ ψ))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breq12:
shows
(A = B $\wedge$ C = D) $\longrightarrow$ (A < C $\longleftrightarrow$ B < D)
(A = B $\wedge$ C = D) $\longrightarrow$ (A $\leq$ C $\longleftrightarrow$ B $\leq$ D)
⟨*proof*⟩

lemma MMI_necom:

shows A $\neq$ B $\longleftrightarrow$ B $\neq$ A
$\langle proof \rangle$

lemma MMI_3jaoi: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch $\longrightarrow \psi$ **and**
A3: $\vartheta \longrightarrow \psi$
shows ($\varphi \vee$ ch $\vee \vartheta$) $\longrightarrow \psi$
$\langle proof \rangle$

lemma MMI_jctr: **assumes** A1: ψ
shows $\varphi \longrightarrow$ ($\varphi \wedge \psi$)
$\langle proof \rangle$

lemma MMI_olc:
shows $\varphi \longrightarrow$ ($\psi \vee \varphi$)
$\langle proof \rangle$

lemma MMI_3syl: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\psi \longrightarrow$ ch **and**
A3: ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow \vartheta$
$\langle proof \rangle$

lemma MMI_mtbird: **assumes** Amin: $\varphi \longrightarrow \neg$ (ch) **and**
Amaj: $\varphi \longrightarrow$ ($\psi \longleftrightarrow$ ch)
shows $\varphi \longrightarrow \neg$ (ψ)
$\langle proof \rangle$

lemma MMI_pm2_21d: **assumes** A1: $\varphi \longrightarrow \neg$ (ψ)
shows $\varphi \longrightarrow$ ($\psi \longrightarrow$ ch)
$\langle proof \rangle$

lemma MMI_3jaodan: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch **and**
A2: ($\varphi \wedge \vartheta$) $\longrightarrow$ ch **and**
A3: ($\varphi \wedge \tau$) $\longrightarrow$ ch
shows ($\varphi \wedge$ ($\psi \vee \vartheta \vee \tau$)) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_sylan2br: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch **and**
A2: $\psi \longleftrightarrow \vartheta$
shows ($\varphi \wedge \vartheta$) $\longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_3jaoian: **assumes** A1: ($\varphi \wedge \psi$) $\longrightarrow$ ch **and**
A2: ($\vartheta \wedge \psi$) $\longrightarrow$ ch **and**
A3: ($\tau \wedge \psi$) $\longrightarrow$ ch
shows (($\varphi \vee \vartheta \vee \tau$) $\wedge \psi$) $\longrightarrow$ ch

⟨*proof*⟩

lemma MMI_mtbid: **assumes** Amin: $\varphi \longrightarrow \neg (\psi)$ **and**
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow ch)$
shows $\varphi \longrightarrow \neg (ch)$
⟨*proof*⟩

lemma MMI_con1d: **assumes** A1: $\varphi \longrightarrow (\neg (\psi) \longrightarrow ch)$
shows $\varphi \longrightarrow (\neg (ch) \longrightarrow \psi)$
⟨*proof*⟩

lemma MMI_pm2_21nd: **assumes** A1: $\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow (\neg (\psi) \longrightarrow ch)$
⟨*proof*⟩

lemma MMI_syl3an1b: **assumes** A1: $(\varphi \wedge \psi \wedge ch) \longrightarrow \vartheta$ **and**
A2: $\tau \longleftrightarrow \varphi$
shows $(\tau \wedge \psi \wedge ch) \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_adantld: **assumes** A1: $\varphi \longrightarrow (\psi \longrightarrow ch)$
shows $\varphi \longrightarrow ((\vartheta \wedge \psi) \longrightarrow ch)$
⟨*proof*⟩

lemma MMI_adantrd: **assumes** A1: $\varphi \longrightarrow (\psi \longrightarrow ch)$
shows $\varphi \longrightarrow ((\psi \wedge \vartheta) \longrightarrow ch)$
⟨*proof*⟩

lemma MMI_anasss: **assumes** A1: $((\varphi \wedge \psi) \wedge ch) \longrightarrow \vartheta$
shows $(\varphi \wedge (\psi \wedge ch)) \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_syl3an3b: **assumes** A1: $(\varphi \wedge \psi \wedge ch) \longrightarrow \vartheta$ **and**
A2: $\tau \longleftrightarrow ch$
shows $(\varphi \wedge \psi \wedge \tau) \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_mpbid: **assumes** Amin: $\varphi \longrightarrow \psi$ **and**
Amaj: $\varphi \longrightarrow (\psi \longleftrightarrow ch)$
shows $\varphi \longrightarrow ch$
⟨*proof*⟩

lemma MMI_orbi12d: **assumes** A1: $\varphi \longrightarrow (\psi \longleftrightarrow ch)$ **and**
A2: $\varphi \longrightarrow (\vartheta \longleftrightarrow \tau)$
shows $\varphi \longrightarrow ((\psi \vee \vartheta) \longleftrightarrow (ch \vee \tau))$
⟨*proof*⟩

lemma MMI_ianor:
  shows $\neg$ (φ $\wedge$ ψ) $\longleftrightarrow$ $\neg$ φ $\vee$ $\neg$ ψ
  ⟨*proof*⟩

lemma MMI_bitr2: **assumes** A1: φ $\longleftrightarrow$ ψ **and**
  A2: ψ $\longleftrightarrow$ ch
  shows ch $\longleftrightarrow$ φ
  ⟨*proof*⟩

lemma MMI_biimp: **assumes** A1: φ $\longleftrightarrow$ ψ
  shows φ $\longrightarrow$ ψ
  ⟨*proof*⟩

lemma MMI_mpan2d: **assumes** A1: φ $\longrightarrow$ ch **and**
  A2: φ $\longrightarrow$ ((ψ $\wedge$ ch) $\longrightarrow$ ϑ)
  shows φ $\longrightarrow$ (ψ $\longrightarrow$ ϑ)
  ⟨*proof*⟩

lemma MMI_ad2antrr: **assumes** A1: φ $\longrightarrow$ ψ
  shows ((φ $\wedge$ ch) $\wedge$ ϑ) $\longrightarrow$ ψ
  ⟨*proof*⟩

lemma MMI_biimpac: **assumes** A1: φ $\longrightarrow$ (ψ $\longleftrightarrow$ ch)
  shows (ψ $\wedge$ φ) $\longrightarrow$ ch
  ⟨*proof*⟩

lemma MMI_con2bii: **assumes** A1: φ $\longleftrightarrow$ $\neg$ (ψ)
  shows ψ $\longleftrightarrow$ $\neg$ (φ)
  ⟨*proof*⟩

lemma MMI_pm3_26bd: **assumes** A1: φ $\longleftrightarrow$ (ψ $\wedge$ ch)
  shows φ $\longrightarrow$ ψ
  ⟨*proof*⟩

lemma MMI_biimpr: **assumes** A1: φ $\longleftrightarrow$ ψ
  shows ψ $\longrightarrow$ φ
  ⟨*proof*⟩

lemma (**in** MMIsar0) MMI_3brtr3g: **assumes** A1: φ $\longrightarrow$ A < B **and**
  A2: A = C **and**
  A3: B = D
  shows φ $\longrightarrow$ C < D
  ⟨*proof*⟩

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
⟨*proof*⟩

lemma MMI_negbii: **assumes** Aa: $\varphi \longleftrightarrow \psi$
shows $\neg\varphi \longleftrightarrow \neg\psi$
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breq1i: **assumes** A1: A = B
shows
A < C $\longleftrightarrow$ B < C
A $\leq$ C $\longleftrightarrow$ B $\leq$ C
⟨*proof*⟩

lemma MMI_syl5eqr: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: A = C
shows $\varphi \longrightarrow$ C = B
⟨*proof*⟩

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
⟨*proof*⟩

lemma MMI_ccase: **assumes** A1: $\varphi \land \psi \longrightarrow \tau$ **and**
A2: ch $\land \psi \longrightarrow \tau$ **and**
A3: $\varphi \land \vartheta \longrightarrow \tau$ **and**
A4: ch $\land \vartheta \longrightarrow \tau$
shows ($\varphi \lor$ ch) $\land$ ($\psi \lor \vartheta$) $\longrightarrow \tau$
⟨*proof*⟩

lemma MMI_pm3_27bd: **assumes** A1: $\varphi \longleftrightarrow \psi \land$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_nsyl3: **assumes** A1: $\varphi \longrightarrow \neg\psi$ **and**
A2: ch $\longrightarrow \psi$
shows ch $\longrightarrow \neg\varphi$
⟨*proof*⟩

lemma MMI_jctild: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**

A2: $\varphi \longrightarrow \vartheta$
shows $\varphi \longrightarrow$
$\psi \longrightarrow \vartheta \wedge$ ch
⟨*proof*⟩

lemma MMI_jctird: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**
A2: $\varphi \longrightarrow \vartheta$
shows $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\wedge \vartheta$
⟨*proof*⟩

lemma MMI_ccase2: **assumes** A1: $\varphi \wedge \psi \longrightarrow \tau$ **and**
A2: ch $\longrightarrow \tau$ **and**
A3: $\vartheta \longrightarrow \tau$
shows ($\varphi \vee$ ch) $\wedge$ ($\psi \vee \vartheta$) $\longrightarrow \tau$
⟨*proof*⟩

lemma MMI_3bitr3r: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**
A2: $\varphi \longleftrightarrow$ ch **and**
A3: $\psi \longleftrightarrow \vartheta$
shows $\vartheta \longleftrightarrow$ ch
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_syl6breq: **assumes** A1: $\varphi \longrightarrow$ A < B **and**
A2: B = C
shows
$\varphi \longrightarrow$ A < C
⟨*proof*⟩

lemma MMI_pm2_61i: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\neg\varphi \longrightarrow \psi$
shows ψ
⟨*proof*⟩

lemma MMI_syl6req: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: B = C
shows $\varphi \longrightarrow$ C = A
⟨*proof*⟩

lemma MMI_pm2_61d: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**
A2: $\varphi \longrightarrow$
$\neg\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_orim1d: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \lor \vartheta \longrightarrow$ ch $\lor \vartheta$
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

lemma MMI_bibi2d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
($\vartheta \longleftrightarrow \psi$) $\longleftrightarrow$
$\vartheta \longleftrightarrow$ ch
⟨*proof*⟩

lemma MMI_con4bid: **assumes** A1: $\varphi \longrightarrow$
$\neg\psi \longleftrightarrow \neg$ch
shows $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
⟨*proof*⟩

lemma MMI_3com13: **assumes** A1: $\varphi \land \psi \land$ ch $\longrightarrow \vartheta$
shows ch $\land \psi \land \varphi \longrightarrow \vartheta$
⟨*proof*⟩

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 $\varphi \longrightarrow$
$\tau \longleftrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_3imtr4g: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**

A2: $\vartheta \longleftrightarrow \psi$ **and**
A3: $\tau \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\vartheta \longrightarrow \tau$
⟨*proof*⟩

lemma MMI_expcom: **assumes** A1: $\varphi \wedge \psi \longrightarrow$ ch
shows $\psi \longrightarrow \varphi \longrightarrow$ ch
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breq2i: **assumes** A1: A = B
shows
C < A $\longleftrightarrow$ C < B
C $\leq$ A $\longleftrightarrow$ C $\leq$ B
⟨*proof*⟩

lemma MMI_3bitr2r: **assumes** A1: $\varphi \longleftrightarrow \psi$ **and**
A2: ch $\longleftrightarrow \psi$ **and**
A3: ch $\longleftrightarrow \vartheta$
shows $\vartheta \longleftrightarrow \varphi$
⟨*proof*⟩

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 $\wedge$ $\vartheta \longrightarrow \tau$
⟨*proof*⟩

lemma MMI_anbi1d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \wedge \vartheta \longleftrightarrow$ ch $\wedge$ ϑ
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breqtrrd: **assumes** A1: $\varphi \longrightarrow$ A < B **and**
A2: $\varphi \longrightarrow$ C = B
shows $\varphi \longrightarrow$ A < C
⟨*proof*⟩

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 $\tau \wedge \eta \wedge \zeta \longrightarrow \vartheta$
⟨*proof*⟩

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 $\varphi \longrightarrow$
$\psi \longleftrightarrow \tau$
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breqtr: **assumes** A1: A < B **and**
A2: B = C
shows A < C
⟨*proof*⟩

lemma MMI_mpi: **assumes** A1: ψ **and**
A2: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_eqtr2: **assumes** A1: A = B **and**
A2: B = C
shows C = A
⟨*proof*⟩

lemma MMI_eqneqi: **assumes** A1: A = B $\longleftrightarrow$ C = D
shows A $\neq$ B $\longleftrightarrow$ C $\neq$ D
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_eqbrtrrd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\varphi \longrightarrow$ A < C
shows $\varphi \longrightarrow$ B < C
⟨*proof*⟩

lemma MMI_mpd: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_mpdan: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \wedge \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_breqtrd: **assumes** A1: $\varphi \longrightarrow$ A < B **and**
A2: $\varphi \longrightarrow$ B = C
shows $\varphi \longrightarrow$ A < C
⟨*proof*⟩

lemma MMI_mpand: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_imbi1d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
($\psi \longrightarrow \vartheta$) $\longleftrightarrow$
(ch $\longrightarrow \vartheta$)
⟨*proof*⟩

lemma MMI_mtbii: **assumes** Amin: $\neg\psi$ **and**
Amaj: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch
shows $\varphi \longrightarrow \neg$ch
⟨*proof*⟩

lemma MMI_sylan2d: **assumes** A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$ **and**
A2: $\varphi \longrightarrow \tau \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \wedge \tau \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_imp32: **assumes** A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

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
⟨*proof*⟩

lemma MMI_a1dd: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \longrightarrow \vartheta \longrightarrow$ ch
⟨*proof*⟩

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
⟨*proof*⟩

lemma MMI_ad2antll: **assumes** A1: $\varphi \longrightarrow \psi$
shows ch $\wedge\ \vartheta \wedge \varphi \longrightarrow \psi$
⟨*proof*⟩

lemma MMI_adantrrl: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge \tau \wedge$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_syl2ani: **assumes** A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$ **and**
A2: $\tau \longrightarrow \psi$ **and**
A3: $\eta \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\tau \wedge \eta \longrightarrow \vartheta$
⟨*proof*⟩

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 $\wedge\ \eta$
⟨*proof*⟩

lemma MMI_ancomsd: **assumes** A1: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$
ch $\wedge\ \psi \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_mpani: **assumes** A1: ψ **and**
A2: $\varphi \longrightarrow$
$\psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \longrightarrow$ ch $\longrightarrow \vartheta$

$\langle proof \rangle$

lemma MMI_syldan: **assumes** A1: $\varphi \wedge \psi \longrightarrow$ ch **and**
A2: $\varphi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \longrightarrow \vartheta$
$\langle proof \rangle$

lemma MMI_mp3anl1: **assumes** A1: φ **and**
A2: ($\varphi \wedge \psi \wedge$ ch) $\wedge \vartheta \longrightarrow \tau$
shows ($\psi \wedge$ ch) $\wedge \vartheta \longrightarrow \tau$
$\langle proof \rangle$

lemma MMI_3ad2ant1: **assumes** A1: $\varphi \longrightarrow$ ch
shows $\varphi \wedge \psi \wedge \vartheta \longrightarrow$ ch
$\langle proof \rangle$

lemma MMI_pm3_2:
shows $\varphi \longrightarrow$
$\psi \longrightarrow \varphi \wedge \psi$
$\langle proof \rangle$

lemma MMI_pm2_43i: **assumes** A1: $\varphi \longrightarrow$
$\varphi \longrightarrow \psi$
shows $\varphi \longrightarrow \psi$
$\langle proof \rangle$

lemma MMI_jctil: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch
shows $\varphi \longrightarrow$ ch $\wedge \psi$
$\langle proof \rangle$

lemma MMI_mpanl12: **assumes** A1: φ **and**
A2: ψ **and**
A3: ($\varphi \wedge \psi$) $\wedge$ ch $\longrightarrow \vartheta$
shows ch $\longrightarrow \vartheta$
$\langle proof \rangle$

lemma MMI_mpanr1: **assumes** A1: ψ **and**
A2: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge$ ch $\longrightarrow \vartheta$
$\langle proof \rangle$

lemma MMI_ad2antrl: **assumes** A1: $\varphi \longrightarrow \psi$
shows ch $\wedge \varphi \wedge \vartheta \longrightarrow \psi$
$\langle proof \rangle$

lemma MMI_3adant3r: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge$ ch $\wedge\ \tau \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_3adant1l: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows ($\tau \wedge \varphi$) $\wedge\ \psi \wedge$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_3adant2r: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge$ ($\psi \wedge \tau$) $\wedge$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_3bitr4rd: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \psi$ **and**
A3: $\varphi \longrightarrow$
$\tau \longleftrightarrow$ ch
shows $\varphi \longrightarrow$
$\tau \longleftrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_3anrev:
shows $\varphi \wedge \psi \wedge$ ch $\longleftrightarrow$ ch $\wedge\ \psi \wedge \varphi$
⟨*proof*⟩

lemma MMI_eqtr4: **assumes** A1: A = B **and**
A2: C = B
shows A = C
⟨*proof*⟩

lemma MMI_anidm:
shows $\varphi \wedge \varphi \longleftrightarrow \varphi$
⟨*proof*⟩

lemma MMI_bi2anan9r: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\vartheta \longrightarrow$
$\tau \longleftrightarrow \eta$
shows $\vartheta \wedge \varphi \longrightarrow$
$\psi \wedge \tau \longleftrightarrow$ ch $\wedge\ \eta$
⟨*proof*⟩

lemma MMI_3imtr3g: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch **and**
A2: $\psi \longleftrightarrow \vartheta$ **and**
A3: ch $\longleftrightarrow \tau$

shows $\varphi \longrightarrow$
$\vartheta \longrightarrow \tau$
$\langle proof \rangle$

lemma MMI_a3d: **assumes** A1: $\varphi \longrightarrow$
$\neg\psi \longrightarrow \neg$ch
shows $\varphi \longrightarrow$ ch $\longrightarrow \psi$
$\langle proof \rangle$

lemma MMI_sylan9bbr: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\vartheta \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\vartheta \wedge \varphi \longrightarrow$
$\psi \longleftrightarrow \tau$
$\langle proof \rangle$

lemma MMI_sylan9bb: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\vartheta \longrightarrow$
ch $\longleftrightarrow \tau$
shows $\varphi \wedge \vartheta \longrightarrow$
$\psi \longleftrightarrow \tau$
$\langle proof \rangle$

lemma MMI_3bitr3g: **assumes** A1: $\varphi \longrightarrow$
$\psi \longleftrightarrow$ ch **and**
A2: $\psi \longleftrightarrow \vartheta$ **and**
A3: ch $\longleftrightarrow \tau$
shows $\varphi \longrightarrow$
$\vartheta \longleftrightarrow \tau$
$\langle proof \rangle$

lemma MMI_pm5_21:
shows $\neg\varphi \wedge \neg\psi \longrightarrow$
$\varphi \longleftrightarrow \psi$
$\langle proof \rangle$

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 $\wedge\ \eta$
$\langle proof \rangle$

lemma MMI_syl3anl1: **assumes** A1: ($\varphi \wedge \psi \wedge$ ch) $\wedge\ \vartheta \longrightarrow \tau$ **and**
A2: $\eta \longrightarrow \varphi$
shows ($\eta \wedge \psi \wedge$ ch) $\wedge\ \vartheta \longrightarrow \tau$
$\langle proof \rangle$

lemma MMI_imp4a: **assumes** A1: $\varphi \longrightarrow$
$\psi \longrightarrow$
ch $\longrightarrow$
$\vartheta \longrightarrow \tau$
shows $\varphi \longrightarrow$
$\psi \longrightarrow$
ch $\wedge$ $\vartheta \longrightarrow \tau$
⟨*proof*⟩

lemma **(in** MMIsar0**)** MMI_breqan12rd: **assumes** A1: $\varphi \longrightarrow$ A = B **and**
A2: $\psi \longrightarrow$ C = D
shows
$\psi \wedge \varphi \longrightarrow$ A < C $\longleftrightarrow$ B < D
$\psi \wedge \varphi \longrightarrow$ A $\leq$ C $\longleftrightarrow$ B $\leq$ D
⟨*proof*⟩

lemma **(in** MMIsar0**)** MMI_3brtr4d: **assumes** A1: $\varphi \longrightarrow$ A < B **and**
A2: $\varphi \longrightarrow$ C = A **and**
A3: $\varphi \longrightarrow$ D = B
shows $\varphi \longrightarrow$ C < D
⟨*proof*⟩

lemma MMI_adantrrr: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge \psi \wedge$ ch $\wedge\ \tau \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_adantrlr: **assumes** A1: $\varphi \wedge \psi \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge (\psi \wedge \tau) \wedge$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_imdistani: **assumes** A1: $\varphi \longrightarrow \psi \longrightarrow$ ch
shows $\varphi \wedge \psi \longrightarrow \varphi \wedge$ ch
⟨*proof*⟩

lemma MMI_anabss3: **assumes** A1: $(\varphi \wedge \psi) \wedge \psi \longrightarrow$ ch
shows $\varphi \wedge \psi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_mp3anl2: **assumes** A1: ψ **and**
A2: $(\varphi \wedge \psi \wedge$ ch$) \wedge \vartheta \longrightarrow \tau$
shows $(\varphi \wedge$ ch$) \wedge \vartheta \longrightarrow \tau$
⟨*proof*⟩

lemma MMI_mpanl2: **assumes** A1: ψ **and**

A2: $(\varphi \wedge \psi) \wedge$ ch $\longrightarrow \vartheta$
shows $\varphi \wedge$ ch $\longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_mpancom: **assumes** A1: $\psi \longrightarrow \varphi$ **and**
A2: $\varphi \wedge \psi \longrightarrow$ ch
shows $\psi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_or12:
shows $\varphi \vee \psi \vee$ ch $\longleftrightarrow \psi \vee \varphi \vee$ ch
⟨*proof*⟩

lemma MMI_rcla4ev: **assumes** A1: $\forall$x. x = A $\longrightarrow$ φ(x) $\longleftrightarrow \psi$
shows A $\in$ B $\wedge \psi \longrightarrow$ ($\exists$x$\in$B. φ(x))
⟨*proof*⟩

lemma MMI_jctir: **assumes** A1: $\varphi \longrightarrow \psi$ **and**
A2: ch
shows $\varphi \longrightarrow \psi \wedge$ ch
⟨*proof*⟩

lemma MMI_iffalse:
shows $\neg\varphi \longrightarrow$ if(φ, A, B) = B
⟨*proof*⟩

lemma MMI_iftrue:
shows $\varphi \longrightarrow$ if(φ, A, B) = A
⟨*proof*⟩

lemma MMI_pm2_61d2: **assumes** A1: $\varphi \longrightarrow$
$\neg\psi \longrightarrow$ ch **and**
A2: $\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_pm2_61dan: **assumes** A1: $\varphi \wedge \psi \longrightarrow$ ch **and**
A2: $\varphi \wedge \neg\psi \longrightarrow$ ch
shows $\varphi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_orcanai: **assumes** A1: $\varphi \longrightarrow \psi \vee$ ch
shows $\varphi \wedge \neg\psi \longrightarrow$ ch
⟨*proof*⟩

lemma MMI_ifcl:
shows A $\in$ C $\wedge$ B $\in$ C $\longrightarrow$ if(φ, A, B) $\in$ C
⟨*proof*⟩

lemma MMI_imim2i: **assumes** A1: $\varphi \longrightarrow \psi$
shows (ch $\longrightarrow \varphi$) $\longrightarrow$ ch $\longrightarrow \psi$
⟨*proof*⟩

lemma MMI_com13: **assumes** A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$
shows ch $\longrightarrow$
$\psi \longrightarrow$
$\varphi \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_rcla4v: **assumes** A1: $\forall$x. x = A $\longrightarrow$ φ(x) $\longleftrightarrow \psi$
shows A $\in$ B $\longrightarrow$ ($\forall$x$\in$B. φ(x)) $\longrightarrow \psi$
⟨*proof*⟩

lemma MMI_syl5d: **assumes** A1: $\varphi \longrightarrow$
$\psi \longrightarrow$ ch $\longrightarrow \vartheta$ **and**
A2: $\varphi \longrightarrow \tau \longrightarrow$ ch
shows $\varphi \longrightarrow$
$\psi \longrightarrow$
$\tau \longrightarrow \vartheta$
⟨*proof*⟩

lemma MMI_eqcoms: **assumes** A1: A = B $\longrightarrow \varphi$
shows B = A $\longrightarrow \varphi$
⟨*proof*⟩

lemma MMI_rgen: **assumes** A1: $\forall$x. x $\in$ A $\longrightarrow$ φ(x)
shows $\forall$x$\in$A. φ(x)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_reex:
shows $\mathbb{R}$ = $\mathbb{R}$
⟨*proof*⟩

lemma MMI_sstri: **assumes** A1: A $\subseteq$B **and**
A2: B $\subseteq$C
shows A $\subseteq$C
⟨*proof*⟩

lemma MMI_ssexi: **assumes** A1: B = B **and**
A2: A $\subseteq$B
shows A = A
⟨*proof*⟩

end

98 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 $A \in \mathbb{R} \longrightarrow A \in \mathbb{C}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_recn: **assumes** A1: $A \in \mathbb{R}$
shows $A \in \mathbb{C}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_recnd: **assumes** A1: $\varphi \longrightarrow A \in \mathbb{R}$
shows $\varphi \longrightarrow A \in \mathbb{C}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_elimne0:
shows if ($A \neq \mathbf{0}$, A , $\mathbf{1}$) $\neq \mathbf{0}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_addex:
shows + isASet
⟨*proof*⟩

lemma (in MMIsar0) MMI_mulex:
shows · isASet
⟨*proof*⟩

lemma (in MMIsar0) MMI_adddirt:
shows ($A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}$) $\longrightarrow$
((A + B) · C) = ((A · C) + (B · C))
⟨*proof*⟩

lemma (in MMIsar0) MMI_addcl: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$
shows (A + B) $\in \mathbb{C}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_mulcl: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$
shows (A · B) $\in \mathbb{C}$
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_addcom: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (A + B) = (B + A)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mulcom: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (A · B) = (B · A)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma (**in** MMIsar0) MMI_1cn:
shows **1** ∈ ℂ
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_0cn:
shows **0** ∈ ℂ
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_addid1: **assumes** A1: A ∈ ℂ
shows (A + **0**) = A
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_addid2: **assumes** A1: A ∈ ℂ
shows (**0** + A) = A

⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mulid1: **assumes** A1: A $\in$ $\mathbb{C}$
shows (A $\cdot$ **1**) = A
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mulid2: **assumes** A1: A $\in$ $\mathbb{C}$
shows (**1** $\cdot$ A) = A
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_negex: **assumes** A1: A $\in$ $\mathbb{C}$
shows $\exists$ x $\in$ $\mathbb{C}$. (A + x) = **0**
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_recex: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: A $\neq$ **0**
shows $\exists$ x $\in$ $\mathbb{C}$. (A $\cdot$ x) = **1**
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_readdcl: **assumes** A1: A $\in$ $\mathbb{R}$ **and**
A2: B $\in$ $\mathbb{R}$
shows (A + B) $\in$ $\mathbb{R}$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_remulcl: **assumes** A1: A $\in$ $\mathbb{R}$ **and**
A2: B $\in$ $\mathbb{R}$
shows (A $\cdot$ B) $\in$ $\mathbb{R}$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_addcan: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$ **and**
A3: C $\in$ $\mathbb{C}$
shows (A + B) = (A + C) $\longleftrightarrow$ B = C
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_addcan2: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$ **and**
A3: C $\in$ $\mathbb{C}$
shows (A + C) = (B + C) $\longleftrightarrow$ A = B
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_addcant:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A + B) = (A + C) $\longleftrightarrow$ B = C)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_addcan2t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$ ((A + C) = (B + C) $\longleftrightarrow$

A = B)
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add12t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$ (A + (B + C)) =
(B + (A + C))
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add23t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$ ((A + B) + C) =
((A + C) + B)
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add4t:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ (C $\in$ $\mathbb{C}$ $\wedge$ D $\in$ $\mathbb{C}$)) $\longrightarrow$
((A + B) + (C + D)) = ((A + C) + (B + D))
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add42t:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ (C $\in$ $\mathbb{C}$ $\wedge$ D $\in$ $\mathbb{C}$)) $\longrightarrow$
((A + B) + (C + D)) = ((A + C) + (D + B))
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add12: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: B $\in$ $\mathbb{C}$ **and**
   A3: C $\in$ $\mathbb{C}$
  shows (A + (B + C)) = (B + (A + C))
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add23: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: B $\in$ $\mathbb{C}$ **and**
   A3: C $\in$ $\mathbb{C}$
  shows ((A + B) + C) = ((A + C) + B)
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add4: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: B $\in$ $\mathbb{C}$ **and**
   A3: C $\in$ $\mathbb{C}$ **and**
   A4: D $\in$ $\mathbb{C}$
  shows ((A + B) + (C + D)) =
((A + C) + (B + D))
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_add42: **assumes** A1: A $\in$ $\mathbb{C}$ **and**

A2: $B \in \mathbb{C}$ **and**
A3: $C \in \mathbb{C}$ **and**
A4: $D \in \mathbb{C}$
shows ((A + B) + (C + D)) =
((A + C) + (D + B))
⟨*proof*⟩

lemma (in MMIsar0) MMI_addid2t:
shows $A \in \mathbb{C} \longrightarrow$ (**0** + A) = A
⟨*proof*⟩

lemma (in MMIsar0) MMI_peano2cn:
shows $A \in \mathbb{C} \longrightarrow$ (A + **1**) $\in \mathbb{C}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_peano2re:
shows $A \in \mathbb{R} \longrightarrow$ (A + **1**) $\in \mathbb{R}$
⟨*proof*⟩

lemma (in MMIsar0) MMI_negeu: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$
shows $\exists!$ x . $x \in \mathbb{C} \wedge$ (A + x) = B
⟨*proof*⟩

lemma (in MMIsar0) MMI_subval: **assumes** $A \in \mathbb{C}$ $B \in \mathbb{C}$
shows A - B = $\bigcup$ { $x \in \mathbb{C}$. B + x = A }
⟨*proof*⟩

lemma (in MMIsar0) MMI_df_neg: **shows** (- A) = **0** - A
⟨*proof*⟩

lemma (in MMIsar0) MMI_negeq:
shows A = B $\longrightarrow$ (-A) = (- B)
⟨*proof*⟩

lemma (in MMIsar0) MMI_negeqi: **assumes** A1: A = B
shows (- A) = (-B)
⟨*proof*⟩

lemma (in MMIsar0) MMI_negeqd: **assumes** A1: $\varphi \longrightarrow$ A = B
shows $\varphi \longrightarrow$ (-A) = (-B)

⟨*proof*⟩

lemma **(in** MMIsar0) MMI_hbneg: **assumes** A1: y ∈ A ⟶ (∀ x . y ∈ A)

shows y ∈ ((- A)) ⟶ (∀ x . (y ∈ ((- A))))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_minusex:
shows ((- A)) isASet ⟨*proof*⟩

lemma **(in** MMIsar0) MMI_subcl: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (A - B) ∈ ℂ
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_subclt:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A - B) ∈ ℂ
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_negclt:
shows A ∈ ℂ ⟶ ((- A)) ∈ ℂ
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_negcl: **assumes** A1: A ∈ ℂ
shows ((- A)) ∈ ℂ
⟨*proof*⟩

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*⟩

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*⟩

lemma **(in** MMIsar0) MMI_subaddt:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A - B) = C ⟷
(B + C) = A)

⟨proof⟩

lemma (**in** MMIsar0) MMI_pncan3t:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + (B - A)) = B
⟨proof⟩

lemma (**in** MMIsar0) MMI_pncan3: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows (A + (B - A)) = B
⟨proof⟩

lemma (**in** MMIsar0) MMI_negidt:
  shows A ∈ ℂ ⟶ (A + ((- A))) = **0**
⟨proof⟩

lemma (**in** MMIsar0) MMI_negid: **assumes** A1: A ∈ ℂ
  shows (A + ((- A))) = **0**
⟨proof⟩

lemma (**in** MMIsar0) MMI_negsub: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows (A + ((- B))) = (A - B)
⟨proof⟩

lemma (**in** MMIsar0) MMI_negsubt:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A + ((- B))) = (A - B)
⟨proof⟩

lemma (**in** MMIsar0) MMI_addsubasst:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
  (A + (B - C))
⟨proof⟩

lemma (**in** MMIsar0) MMI_addsubt:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A + B) - C) =
  ((A - C) + B)
⟨proof⟩

lemma (**in** MMIsar0) MMI_addsub12t:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A + (B - C)) =
  (B + (A - C))
⟨proof⟩

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⟩

lemma **(in** MMIsar0) MMI_addsub: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $C \in \mathbb{C}$
shows ((A + B) - C) = ((A - C) + B)
⟨proof⟩

lemma **(in** MMIsar0) MMI_2addsubt:
shows (($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\wedge$ ($C \in \mathbb{C} \wedge D \in \mathbb{C}$)) $\longrightarrow$
(((A + B) + C) - D) = (((A + C) - D) + B)
⟨proof⟩

lemma **(in** MMIsar0) MMI_negneg: **assumes** A1: $A \in \mathbb{C}$
shows (- ((- A))) = A
⟨proof⟩

lemma **(in** MMIsar0) MMI_subid: **assumes** A1: $A \in \mathbb{C}$
shows (A - A) = **0**
⟨proof⟩

lemma **(in** MMIsar0) MMI_subid1: **assumes** A1: $A \in \mathbb{C}$
shows (A - **0**) = A
⟨proof⟩

lemma **(in** MMIsar0) MMI_negnegt:
shows $A \in \mathbb{C} \longrightarrow$ (- ((- A))) = A
⟨proof⟩

lemma **(in** MMIsar0) MMI_subnegt:
shows ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ (A - ((- B))) = (A + B)
⟨proof⟩

lemma **(in** MMIsar0) MMI_subidt:
shows $A \in \mathbb{C} \longrightarrow$ (A - A) = **0**
⟨proof⟩

lemma **(in** MMIsar0) MMI_subid1t:
shows $A \in \mathbb{C} \longrightarrow$ (A - **0**) = A
⟨proof⟩

lemma **(in** MMIsar0) MMI_pncant:
shows ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ ((A + B) - B) = A
⟨proof⟩

lemma **(in** MMIsar0) MMI_pncan2t:
shows ($A \in \mathbb{C} \wedge B \in \mathbb{C}$) $\longrightarrow$ ((A + B) - A) = B
⟨proof⟩

lemma **(in** MMIsar0) MMI_npcant:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A - B) + B) = A
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_npncant:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
  ((A - B) + (B - C)) = (A - C)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_nppcant:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
  (((A - B) + C) + B) = (A + C)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_subneg: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows (A - ((- B))) = (A + B)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_subeq0: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows (A - B) = **0** ⟷ A = B
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_neg11: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows ((- A)) = ((- B)) ⟷ A = B
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_negcon1: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows ((- A)) = B ⟷ ((- B)) = A
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_negcon2: **assumes** A1: A ∈ ℂ **and**
  A2: B ∈ ℂ
  shows A = ((- B)) ⟷ B = ((- A))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_neg11t:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (((- A)) = ((- B)) ⟷ A = B)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_negcon1t:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (((- A)) = B ⟷ ((- B)) = A)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negcon2t:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ (A = ((- B)) ⟷ B = ((- A)))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subcant:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A - B) =
 (A - C) ⟷ B = C)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subcan2t:
  shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
 ((A - C) = (B - C) ⟷ A = B)
⟨*proof*⟩

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*⟩

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*⟩

lemma (**in** MMIsar0) MMI_subeq0t:
  shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((A - B) = **0** ⟷ A = B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_neg0:
  shows (- **0**) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_renegcl: **assumes** A1: A ∈ ℝ
  shows ((- A)) ∈ ℝ
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_renegclt:
  shows A ∈ ℝ ⟶ ((- A)) ∈ ℝ
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_resubclt:
  shows (A ∈ ℝ ∧ B ∈ ℝ) ⟶ (A - B) ∈ ℝ
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_resubcl: **assumes** A1: A ∈ ℝ **and**
A2: B ∈ ℝ
shows (A - B) ∈ ℝ
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_0re:
shows **0** ∈ ℝ
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mulid2t:
shows A ∈ ℂ ⟶ (**1** · A) = A
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mul12t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ (A · (B · C)) =
(B · (A · C))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mul23t:
shows (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶ ((A · B) · C) =
((A · C) · B)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mul4t:
shows ((A ∈ ℂ ∧ B ∈ ℂ) ∧ (C ∈ ℂ ∧ D ∈ ℂ)) ⟶
((A · B) · (C · D)) = ((A · C) · (B · D))
⟨*proof*⟩

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*⟩

lemma **(in** MMIsar0) MMI_muladd11t:
shows (A ∈ ℂ ∧ B ∈ ℂ) ⟶ ((**1** + A) · (**1** + B)) =
((**1** + A) + (B + (A · B)))
⟨*proof*⟩

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*⟩

lemma (**in** MMIsar0) MMI_mul23: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$
  shows ((A · B) · C) = ((A · C) · B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul4: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$ **and**
    A4: D $\in$ $\mathbb{C}$
  shows ((A · B) · (C · D)) = ((A · C) · (B · D))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_muladd: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$ **and**
    A4: D $\in$ $\mathbb{C}$
  shows ((A + B) · (C + D)) =
  (((A · C) + (D · B)) + ((A · D) + (C · B)))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subdit:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
 (A · (B - C)) = ((A · B) - (A · C))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subdirt:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
 ((A - B) · C) = ((A · C) - (B · C))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subdi: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$
  shows (A · (B - C)) = ((A · B) - (A · C))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subdir: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
    A2: B $\in$ $\mathbb{C}$ **and**
    A3: C $\in$ $\mathbb{C}$
  shows ((A - B) · C) = ((A · C) - (B · C))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul01: **assumes** A1: A $\in$ $\mathbb{C}$
  shows (A · **0**) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul02: **assumes** A1: A $\in$ $\mathbb{C}$

shows (**0** · A) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_1p1times: **assumes** A1: A ∈ ℂ
shows ((**1** + **1**) · A) = (A + A)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul01t:
shows A ∈ ℂ ⟶ (A · **0**) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul02t:
shows A ∈ ℂ ⟶ (**0** · A) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mulneg1: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (((- A)) · B) = (- (A · B))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mulneg2: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (A · ((- B))) =
(- (A · B))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul2neg: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (((- A)) · ((- B))) =
(A · B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negdi: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (- (A + B)) =
(((- A)) + ((- B)))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negsubdi: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (- (A - B)) =
(((- A)) + B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negsubdi2: **assumes** A1: A ∈ ℂ **and**
A2: B ∈ ℂ
shows (- (A - B)) = (B - A)

⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mulneg1t:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (((- A)) · B) =
 (- (A · B))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mulneg2t:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (A · ((- B))) =
 (- (A · B))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mulneg12t:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (((- A)) · B) =
 (A · ((- B)))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_mul2negt:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (((- A)) · ((- B))) =
 (A · B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negdit:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (- (A + B)) =
 (((- A)) + ((- B)))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negdi2t:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (- (A + B)) = (((- A)) - B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negsubdit:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (- (A - B)) = (((- A)) + B)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_negsubdi2t:
    shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
 (- (A - B)) = (B - A)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_subsub2t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A - (B - C)) = (A + (C - B))
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_subsubt:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A - (B - C)) = ((A - B) + C)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_subsub3t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
(A - (B - C)) = ((A + C) - B)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_subsub4t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A - B) - C) = (A - (B + C))
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_sub23t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A - B) - C) = ((A - C) - B)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_nnncant:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A - (B - C)) - C) = (A - B)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_nnncan1t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A - B) - (A - C)) = (C - B)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_nnncan2t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A - C) - (B - C)) = (A - B)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_nncant:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\longrightarrow$
(A - (A - B)) = B
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_nppcan2t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$

((A - (B + C)) + C) = (A - B)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mulm1t:
  shows A $\in$ $\mathbb{C}$ $\longrightarrow$ ((- **1**) $\cdot$ A) = ((- A))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mulm1: **assumes** A1: A $\in$ $\mathbb{C}$
  shows ((- **1**) $\cdot$ A) = ((- A))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_sub4t:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ (C $\in$ $\mathbb{C}$ $\wedge$ D $\in$ $\mathbb{C}$)) $\longrightarrow$
((A + B) - (C + D)) =
((A - C) + (B - D))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_sub4: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
  A2: B $\in$ $\mathbb{C}$ **and**
  A3: C $\in$ $\mathbb{C}$ **and**
  A4: D $\in$ $\mathbb{C}$
  shows ((A + B) - (C + D)) =
((A - C) + (B - D))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_mulsubt:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$) $\wedge$ (C $\in$ $\mathbb{C}$ $\wedge$ D $\in$ $\mathbb{C}$)) $\longrightarrow$
((A - B) $\cdot$ (C - D)) =
(((A $\cdot$ C) + (D $\cdot$ B)) - ((A $\cdot$ D) + (C $\cdot$ B)))
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_pnpcant:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A + B) - (A + C)) = (B - C)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_pnpcan2t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A + C) - (B + C)) = (A - B)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_pnncant:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$
((A + B) - (A - C)) = (B + C)
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_ppncant:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\longrightarrow$

$((A + B) + (C - B)) = (A + C)$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_pnncan: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $C \in \mathbb{C}$
shows $((A + B) - (A - C)) = (B + C)$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_mulcan: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $C \in \mathbb{C}$ **and**
A4: $A \neq \mathbf{0}$
shows $(A \cdot B) = (A \cdot C) \longleftrightarrow B = C$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_mulcant2: **assumes** A1: $A \neq \mathbf{0}$
shows $(A \in \mathbb{C} \land B \in \mathbb{C} \land C \in \mathbb{C}) \longrightarrow$
$((A \cdot B) = (A \cdot C) \longleftrightarrow B = C)$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_mulcant:
shows $((A \in \mathbb{C} \land B \in \mathbb{C} \land C \in \mathbb{C}) \land A \neq \mathbf{0}) \longrightarrow$
$((A \cdot B) = (A \cdot C) \longleftrightarrow B = C)$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_mulcan2t:
shows $((A \in \mathbb{C} \land B \in \mathbb{C} \land C \in \mathbb{C}) \land C \neq \mathbf{0}) \longrightarrow$
$((A \cdot C) = (B \cdot C) \longleftrightarrow A = B)$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_mul0or: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$
shows $(A \cdot B) = \mathbf{0} \longleftrightarrow (A = \mathbf{0} \lor B = \mathbf{0})$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_msq0: **assumes** A1: $A \in \mathbb{C}$
shows $(A \cdot A) = \mathbf{0} \longleftrightarrow A = \mathbf{0}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_mul0ort:
shows $(A \in \mathbb{C} \land B \in \mathbb{C}) \longrightarrow$
$((A \cdot B) = \mathbf{0} \longleftrightarrow (A = \mathbf{0} \lor B = \mathbf{0}))$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_muln0bt:
shows $(A \in \mathbb{C} \land B \in \mathbb{C}) \longrightarrow$

$((A \neq \mathbf{0} \wedge B \neq \mathbf{0}) \longleftrightarrow (A \cdot B) \neq \mathbf{0})$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_muln0: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $A \neq \mathbf{0}$ **and**
A4: $B \neq \mathbf{0}$
shows $(A \cdot B) \neq \mathbf{0}$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_receu: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $A \neq \mathbf{0}$
shows $\exists ! x . x \in \mathbb{C} \wedge (A \cdot x) = B$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_divval: **assumes** $A \in \mathbb{C}$ $B \in \mathbb{C}$ $B \neq \mathbf{0}$
shows $A / B = \bigcup \{ x \in \mathbb{C} . B \cdot x = A \}$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_divmul: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $C \in \mathbb{C}$ **and**
A4: $B \neq \mathbf{0}$
shows $(A / B) = C \longleftrightarrow (B \cdot C) = A$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_divmulz: **assumes** A1: $A \in \mathbb{C}$ **and**
A2: $B \in \mathbb{C}$ **and**
A3: $C \in \mathbb{C}$
shows $B \neq \mathbf{0} \longrightarrow$
$((A / B) = C \longleftrightarrow (B \cdot C) = A)$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_divmult:
shows $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge B \neq \mathbf{0}) \longrightarrow$
$((A / B) = C \longleftrightarrow (B \cdot C) = A)$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_divmul2t:
shows $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge B \neq \mathbf{0}) \longrightarrow$
$((A / B) = C \longleftrightarrow A = (B \cdot C))$
⟨*proof*⟩

lemma **(in** MMIsar0) MMI_divmul3t:
  shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ B $\neq$ **0**) $\longrightarrow$
((A / B) = C $\longleftrightarrow$ A = (C $\cdot$ B))
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_divcl: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
  A2: B $\in$ $\mathbb{C}$ **and**
  A3: B $\neq$ **0**
  shows (A / B) $\in$ $\mathbb{C}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_divclz: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
  A2: B $\in$ $\mathbb{C}$
  shows B $\neq$ **0** $\longrightarrow$ (A / B) $\in$ $\mathbb{C}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_divclt:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ B $\neq$ **0**) $\longrightarrow$
(A / B) $\in$ $\mathbb{C}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_reccl: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
  A2: A $\neq$ **0**
  shows (**1** / A) $\in$ $\mathbb{C}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_recclz: **assumes** A1: A $\in$ $\mathbb{C}$
  shows A $\neq$ **0** $\longrightarrow$ (**1** / A) $\in$ $\mathbb{C}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_recclt:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$ (**1** / A) $\in$ $\mathbb{C}$
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_divcan2: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
  A2: B $\in$ $\mathbb{C}$ **and**
  A3: A $\neq$ **0**
  shows (A $\cdot$ (B / A)) = B
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_divcan1: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
  A2: B $\in$ $\mathbb{C}$ **and**
  A3: A $\neq$ **0**
  shows ((B / A) $\cdot$ A) = B
$\langle proof \rangle$

lemma **(in** MMIsar0) MMI_divcan1z: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$
shows A $\neq$ **0** $\longrightarrow$ ((B / A) $\cdot$ A) = B
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_divcan2z: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$
shows A $\neq$ **0** $\longrightarrow$ (A $\cdot$ (B / A)) = B
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_divcan1t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
((B / A) $\cdot$ A) = B
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_divcan2t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
(A $\cdot$ (B / A)) = B
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_divne0bt:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ B $\neq$ **0**) $\longrightarrow$
(A $\neq$ **0** $\longleftrightarrow$ (A / B) $\neq$ **0**)
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_divne0: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$ **and**
A3: A $\neq$ **0** **and**
A4: B $\neq$ **0**
shows (A / B) $\neq$ **0**
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_recne0z: **assumes** A1: A $\in$ $\mathbb{C}$
shows A $\neq$ **0** $\longrightarrow$ (**1** / A) $\neq$ **0**
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_recne0t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$ (**1** / A) $\neq$ **0**
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_recid: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: A $\neq$ **0**
shows (A $\cdot$ (**1** / A)) = **1**
$\langle$*proof*$\rangle$

lemma **(in** MMIsar0) MMI_recidz: **assumes** A1: A $\in$ $\mathbb{C}$
shows A $\neq$ **0** $\longrightarrow$ (A $\cdot$ (**1** / A)) = **1**

$\langle proof \rangle$

lemma (in MMIsar0) MMI_recidt:
shows (A $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
(A $\cdot$ (**1** / A)) = **1**
$\langle proof \rangle$

lemma (in MMIsar0) MMI_recid2t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$
((**1** / A) $\cdot$ A) = **1**
$\langle proof \rangle$

lemma (in MMIsar0) MMI_divrec: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$ **and**
A3: B $\neq$ **0**
shows (A / B) = (A $\cdot$ (**1** / B))
$\langle proof \rangle$

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 $\cdot$ (**1** / B))
$\langle proof \rangle$

lemma (in MMIsar0) MMI_divrect:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ B $\neq$ **0**) $\longrightarrow$
(A / B) = (A $\cdot$ (**1** / B))
$\langle proof \rangle$

lemma (in MMIsar0) MMI_divrec2t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ B $\neq$ **0**) $\longrightarrow$
(A / B) = ((**1** / B) $\cdot$ A)
$\langle proof \rangle$

lemma (in MMIsar0) MMI_divasst:
shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ B) / C) = (A $\cdot$ (B / C))
$\langle proof \rangle$

lemma (in MMIsar0) MMI_div23t:
shows ((A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$
((A $\cdot$ B) / C) = ((A / C) $\cdot$ B)
$\langle proof \rangle$

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) $\cdot$ C) = ((C / B) $\cdot$ A)
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_div12t:
  shows $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$(A \cdot (B / C)) = (B \cdot (A / C))$
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_divassz: **assumes** A1: $A \in \mathbb{C}$ **and**
  A2: $B \in \mathbb{C}$ **and**
  A3: $C \in \mathbb{C}$
  shows $C \neq \mathbf{0} \longrightarrow$
$((A \cdot B) / C) = (A \cdot (B / C))$
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_divass: **assumes** A1: $A \in \mathbb{C}$ **and**
  A2: $B \in \mathbb{C}$ **and**
  A3: $C \in \mathbb{C}$ **and**
  A4: $C \neq \mathbf{0}$
  shows $((A \cdot B) / C) = (A \cdot (B / C))$
$\langle proof \rangle$

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: $C \neq \mathbf{0}$
  shows $((A + B) / C) =$
$((A / C) + (B / C))$
$\langle proof \rangle$

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: $C \neq \mathbf{0}$
  shows $((A \cdot B) / C) = ((A / C) \cdot B)$
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_divdirz: **assumes** A1: $A \in \mathbb{C}$ **and**
  A2: $B \in \mathbb{C}$ **and**
  A3: $C \in \mathbb{C}$
  shows $C \neq \mathbf{0} \longrightarrow$
$((A + B) / C) =$
$((A / C) + (B / C))$
$\langle proof \rangle$

lemma (**in** MMIsar0) MMI_divdirt:
  shows $((A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C}) \wedge C \neq \mathbf{0}) \longrightarrow$
$((A + B) / C) =$

((A / C) + (B / C))
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divcan3: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$ **and**
A3: A $\neq$ $\mathbf{0}$
shows ((A $\cdot$ B) / A) = B
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divcan4: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$ **and**
A3: A $\neq$ $\mathbf{0}$
shows ((B $\cdot$ A) / A) = B
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divcan3z: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$
shows A $\neq$ $\mathbf{0}$ $\longrightarrow$ ((A $\cdot$ B) / A) = B
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divcan4z: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
A2: B $\in$ $\mathbb{C}$
shows A $\neq$ $\mathbf{0}$ $\longrightarrow$ ((B $\cdot$ A) / A) = B
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divcan3t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ $\mathbf{0}$) $\longrightarrow$
((A $\cdot$ B) / A) = B
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divcan4t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ $\mathbf{0}$) $\longrightarrow$
((B $\cdot$ A) / A) = B
⟨*proof*⟩

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 $\neq$ $\mathbf{0}$
shows (A / C) = (B / C) $\longleftrightarrow$ A = B
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_div11t:
shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ (C $\in$ $\mathbb{C}$ $\wedge$ C $\neq$ $\mathbf{0}$)) $\longrightarrow$
((A / C) = (B / C) $\longleftrightarrow$ A = B)
⟨*proof*⟩

end

99 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 $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$ (A / A) = **1**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_div0t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ A $\neq$ **0**) $\longrightarrow$ (**0** / A) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_diveq0t:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ C $\in$ $\mathbb{C}$ $\wedge$ C $\neq$ **0**) $\longrightarrow$
 ((A / C) = **0** $\longleftrightarrow$ A = **0**)
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_recrec: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: A $\neq$ **0**
  shows (**1** / (**1** / A)) = A
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divid: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: A $\neq$ **0**
  shows (A / A) = **1**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_div0: **assumes** A1: A $\in$ $\mathbb{C}$ **and**
   A2: A $\neq$ **0**
  shows (**0** / A) = **0**
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_div1: **assumes** A1: A $\in$ $\mathbb{C}$
  shows (A / **1**) = A
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_div1t:
  shows A $\in$ $\mathbb{C}$ $\longrightarrow$ (A / **1**) = A
⟨*proof*⟩

lemma (**in** MMIsar0) MMI_divnegt:
  shows (A $\in$ $\mathbb{C}$ $\wedge$ B $\in$ $\mathbb{C}$ $\wedge$ B $\neq$ **0**) $\longrightarrow$
 (- (A / B)) = ((- A) / B)

⟨*proof*⟩

```
lemma (in MMIsar0) MMI_divsubdirt:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( ( A - B ) / C ) =
 ( ( A / C ) - ( B / C ) )
```

⟨*proof*⟩

end

100 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.

100.1 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*⟩

end

101 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`.

101.1 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*⟩

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*⟩

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*⟩

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⟩
```

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⟩
```

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⟩
```

101.2 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⟩
```

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⟩
```

Some simple arithmetics.

```
lemma (in complex0) arith: shows
  2 + 2 = 4
  2·2 = 4
  3·2 = 6
  3·3 = 9
⟨proof⟩
```

101.3 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⟩
```

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⟩
```

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⟩
```

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⟩

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.