IsarMathLib

Slawomir Kolodynski

November 29, 2006

Abstract

This is the proof document of the IsarMathLib project version 1.3.0. IsarMathLib is a library of formalized mathematics for Isabelle 2005 (ZF logic).

Contents

1 Fol1.thy

theory Fol1 **imports** Trancl

begin

1.1 Mission statement

Until we come up with something better let's just say that writing formalized proofs protects from Alzheimer's disease better than solving crossword puzzles.

1.2 Release notes

This release continues the process of importing Metamath's [4] set.mm database into IsarMathLib, adding about 440 facts and 200 translated proofs. We also add a construction of a model of complex numbers from a complete ordered field.

1.3 Overview of the project

The theory files Fol1, ZF1, Nat_ZF, func1, func_ZF, EquivClass1, Finite1, Finite_ZF, Order_ZF contain some background material that is needed for the remaining theories.

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

Group_ZF, Group_ZF_1, 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.

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

Int_ZF 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.

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

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 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 showing that real numbers constructed this way form a complete ordered field.

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.

The `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.thy` and `MMI_Complex_1` 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 `known_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 looks like.

1.4 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` results up Isabelle in an infinite loop). We reluctantly use `(unfold trans_def)` after the `proof` keyword to workaround this.

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

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

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

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

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

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

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

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

```
lemma Fol1_L5:
  assumes p∨q∨r
  and p ⟶ ¬q ∧ ¬r
  and q ⟶ ¬p ∧ ¬r
  and r ⟶ ¬p ∧ ¬q
  shows Exactly_1_of_3_holds (p,q,r)
proof -
  from prems have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    by blast
  then show Exactly_1_of_3_holds (p,q,r)
    by (unfold Exactly_1_of_3_holds_def)
qed
```

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

```
lemma Fol1_L6:
  assumes A1: ¬p and A2: Exactly_1_of_3_holds (p,q,r)
  shows q∨r
```

```
proof -
  from A2 have
    (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶ ¬p ∧ ¬q)
    by (unfold Exactly_1_of_3_holds_def)
  then have p∨q∨r by blast
  with A1 show q∨r by simp
qed
```

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

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

The next lemma demonstrates an elegant form of the Exactly_1_of_3_holds (p,q,r) predicate. More on that at www.solcon.nl/mklooster/calc/calc-tri.html .

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

A property of the Exactly_1_of_3_holds predicate.

```
lemma Fol1_L8A: assumes A1: Exactly_1_of_3_holds (p,q,r)
  shows p ⟷ ¬(q ∨ r)
proof -
  from A1 have (p∨q∨r) ∧ (p ⟶ ¬q ∧ ¬r) ∧ (q ⟶ ¬p ∧ ¬r) ∧ (r ⟶
¬p ∧ ¬q)
    by (unfold Exactly_1_of_3_holds_def)
  then show p ⟷ ¬(q ∨ r) by blast
qed
```

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

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

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

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

Equivalence relations are symmetric.

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

This lemma is needed to be used as a rule in some very complicated cases.

```
lemma five_more_conj: assumes Axs  Ax1 Ax2 Ax3 Ax4 Ax5
  shows Ax1 ∧ Ax2 ∧ Ax3 ∧ Ax4 ∧ Ax5 ∧ Axs using prems by simp
```

```
end
```

2 ZF1.thy

theory ZF1 **imports** pair

begin

2.1 Lemmas in Zermelo-Fraenkel set theory

Here we put lemmas from the set theory that we could not find in the standard Isabelle distribution.

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

```
lemma ZF1_1_L1: assumes C≠0 and ∀y∈C. b(y) = A
  shows (⋃y∈C. b(y)) = A using prems by blast
```

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

```
lemma ZF1_1_L2: assumes A1:C≠0 and A2: ∀x∈C. b(x) ∈ A
  and A3: ∀x y. x∈C ∧ y∈C ⟶ b(x) = b(y)
  shows (⋃x∈C. b(x))∈A
proof -
  from A1 obtain x where D1:x∈C by auto
  with A3 have ∀y∈C. b(y) = b(x) by blast
  with A1 have (⋃y∈C. b(y)) = b(x)
    using ZF1_1_L1 by simp
  with D1 A2 show thesis by simp
qed
```

A purely technical lemma that shows what it means that something belongs to a subset of cartesian product defined by separation. Seems there is no way to avoid that ugly lambda notation.

```
lemma ZF1_1_L3: assumes A1: x∈X  y∈Y and A2: z = a(x,y)
  shows z ∈ {a(x,y).⟨x,y⟩ ∈ X×Y}
proof
  from A2 show z = (λ ⟨x,y⟩. a(x, y))(<x,y>) by simp
  from A1 show <x,y> ∈ X×Y by simp
qed
```

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

```
lemma ZF1_1_L4: assumes A1: ∀x∈X.∀y∈Y. a(x,y) = b(x,y)
  shows {a(x,y). ⟨x,y⟩ ∈ X×Y} = {b(x,y). ⟨x,y⟩ ∈ X×Y}
proof
  show {a(x, y). ⟨x,y⟩ ∈ X × Y} ⊆ {b(x, y). ⟨x,y⟩ ∈ X × Y}
  proof
    fix z assume z ∈ {a(x, y) . ⟨x,y⟩ ∈ X × Y}
    then obtain x y where T1: z = a(x,y) x∈X y∈Y
      by auto
```

```
    with A1 have z = b(x,y) x∈X y∈Y by simp
    then show  z ∈ {b(x,y).⟨x,y⟩ ∈ X×Y}
      using ZF1_1_L3 by simp
  qed
  show {b(x, y). ⟨x,y⟩ ∈ X × Y} ⊆ {a(x, y). ⟨x,y⟩ ∈ X × Y}
  proof
    fix z assume z ∈ {b(x, y). ⟨x,y⟩ ∈ X × Y}
    then obtain x y where T1: z = b(x,y) x∈X y∈Y
      by auto
    with A1 have z = a(x,y) x∈X y∈Y by simp
    then show z ∈ {a(x,y).⟨x,y⟩ ∈ X×Y}
      using ZF1_1_L3 by simp
  qed
qed
```

If two meta-functions are the same on a cartesian product, then the subsets defined by them are the same. I am surprised blast can not handle this. This is similar to `ZF1_1_L4`, except that the set definition varies over `p∈X×Y` rather than `<x,y>∈X×Y`.

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

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

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

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

```
lemma ZF1_1_L5: assumes X≠0 and ∀x∈X. b(x) = c
  shows {b(x). x∈X} = {c} using prems by blast
```

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

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

We can choose an element from a nonempty set.

```
lemma nonempty_has_element: assumes X≠0 shows ∃x. x∈X
  using prems by auto
```

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

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

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

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

An intersection of subsets is a subset.

```
lemma ZF1_1_L7: assumes A1: I≠0 and A2: ∀i∈I. P(i) ⊆ X
  shows ( ⋂i∈I. P(i) ) ⊆ X
proof -
  from A1 obtain i0 where i0 ∈ I by auto
  with A2 have ( ⋂i∈I. P(i) ) ⊆ P(i0) and P(i0) ⊆ X
    by auto
  thus ( ⋂i∈I. P(i) ) ⊆ X by auto
qed

end
```

3 Nat_ZF.thy

```
theory Nat_ZF imports Nat

begin
```

This theory contains lemmas that are missing from the standard Isabelle's Nat.thy file.

3.1 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 induction step for the first order induction.

```
lemma Nat_ZF_1_L1: assumes x∈nat P(x)
  and ∀k∈nat. P(k)⟶P(succ(k))
  shows P(succ(x)) using prems by simp
```

The actual first order induction on natural numbers.

```
lemma Nat_ZF_1_L2:
  assumes A1: n∈nat and A2: P(0) and A3: ∀k∈nat. P(k)⟶P(succ(k))
  shows P(n)
proof -
  from A1 A2 have n∈nat P(0) by auto
  then show P(n) using Nat_ZF_1_L1 by (rule nat_induct)
qed
```

A nonzero natural number has a predecessor.

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

end
```

4 func1.thy

theory func1 **imports** func Fol1 ZF1

begin

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.

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

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

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

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

A first-order version of Pi_type.

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

There is a value for each argument.

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

Inverse image of any set is contained in the domain.

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

```
    then show thesis by auto
qed
```

The inverse image of the range is the domain.

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

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

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

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

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

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

```
lemma func1_1_L5B:
  assumes  A1:f:X→Y shows range(f) ⊆ Y
proof
  fix y assume y ∈ range(f)
  then obtain x where <x,y> ∈ f
    using range_def converse_def domain_def by auto
  with A1 show y∈Y using func1_1_L5 by blast
qed
```

The image of any set is contained in the range.

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

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

```
lemma func1_1_L6A: assumes A1: f:X→Y shows f-(A)⊆X
proof
  fix x
```

```
  assume A2: x∈f-(A) then obtain y where <x,y> ∈ f
    using vimage_iff by auto
  with A1 show  x∈X using func1_1_L5 by fast
qed
```

Inverse image of a greater set is greater.

```
lemma func1_1_L7: assumes A⊆B and function(f)
  shows f-(A)⊆ f-(B) using prems function_vimage_Diff by auto
```

Image of a greater set is greater.

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

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

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

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

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

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

```
lemma func1_1_L11:
  assumes f ⊆ X×Y and ∀x∈X. ∃!y. y∈Y & <x,y> ∈ f
  shows f: X→Y using prems func1_1_L10 Pi_iff_old by simp
```

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

```
lemma func1_1_L11A: assumes A1: ∀x∈X. b(x)∈Y
  shows {<x,y> ∈ X×Y. b(x) = y} : X→Y
proof -
  let f = {<x,y> ∈ X×Y. b(x) = y}
```

```
  have f ⊆ X×Y by auto
  moreover have ∀x∈X. ∃!y. y∈Y & <x,y> ∈ f
  proof
    fix x assume A2: x∈X
    show ∃!y. y∈Y ∧ ⟨x, y⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
    proof
      def y ≡ b(x)
      with A2 A1 show
        ∃y. y∈Y & ⟨x, y⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
        by simp
    next
      fix y y1
      assume y∈Y ∧ ⟨x, y⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
        and y1∈Y ∧ ⟨x, y1⟩ ∈ {⟨x,y⟩ ∈ X×Y . b(x) = y}
      then show y = y1 by simp
    qed
  qed
  ultimately show {<x,y> ∈ X×Y. b(x) = y} : X→Y
    using func1_1_L11 by simp
qed
```

The next lemma will replace `func1_1_L11A` one day.

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

The value of a function defined by a meta-function is this meta-function.

```
lemma func1_1_L11B:
  assumes A1: f:X→Y   x∈X
  and A2: f = {<x,y> ∈ X×Y. b(x) = y}
  shows f(x) = b(x)
proof -
```

```
  from A1 have <x,f(x)> ∈ f using apply_iff by simp
  with A2 show thesis by simp
qed
```

The next lemma will replace `func1_1_L11B` one day.

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

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 = 0 and Dg : g = f ∪ {⟨x,b(x)⟩. x∈A}
  shows
  g : X∪A → Y∪B
  ∀x∈X. g(x) = f(x)
  ∀x∈A. g(x) = b(x)
proof -
  let h = {⟨x,b(x)⟩. x∈A}
  from A1 A2 A3 have
    I: f:X→Y  h : A→B  X∩A = 0
    using ZF_fun_from_total by auto
  then have f∪h : X∪A → Y∪B
    by (rule fun_disjoint_Un)
  with Dg show g : X∪A → Y∪B by simp
  { fix x assume A4: x∈A
    with A1 A3 have (f∪h)(x) = h(x)
      using func1_1_L1 fun_disjoint_apply2
      by blast
    moreover from I A4 have h(x) = b(x)
      using ZF_fun_from_tot_val by simp
    ultimately have (f∪h)(x) = b(x)
      by simp
  } with Dg show ∀x∈A. g(x) = b(x) by simp
  { fix x assume A5: x∈X
    with A3 I have x ∉ domain(h)
      using func1_1_L1 by auto
    then have (f∪h)(x) = f(x)
      using fun_disjoint_apply1 by simp
  } with Dg show ∀x∈X. g(x) = f(x) by simp
qed
```

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

```
lemma func1_1_L11D: assumes A1: f:X→Y and A2: a∉X
  and Dg: g = f ∪ {⟨a,b⟩}
  shows
  g : X∪{a} → Y∪{b}
  ∀x∈X. g(x) = f(x)
  g(a) = b
proof -
  let h = {⟨a,b⟩}
  from A1 A2 Dg have I:
    f:X→Y  ∀x∈{a}. b∈{b}  X∩{a} = 0  g = f ∪ {⟨x,b⟩. x∈{a}}
    by auto
  then show g : X∪{a} → Y∪{b}
    by (rule func1_1_L11C)
  from I show ∀x∈X. g(x) = f(x)
    by (rule func1_1_L11C)
  from I have ∀x∈{a}. g(x) = b
    by (rule func1_1_L11C)
  then show g(a) = b by auto
qed
```

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

```
lemma func1_1_L11E:
  assumes A1: f:X→Y and
  A2: ∀x∈A. b(x)∈B and
  A3: X∩A = 0 and A4: a∉ X∪A
  and Dg: g = f ∪ {⟨x,b(x)⟩. x∈A} ∪ {⟨a,c⟩}
  shows
  g : X∪A∪{a} → Y∪B∪{c}
  ∀x∈X. g(x) = f(x)
  ∀x∈A. g(x) = b(x)
  g(a) = c
proof -
  let h = f ∪ {⟨x,b(x)⟩. x∈A}
  from prems show g : X∪A∪{a} → Y∪B∪{c}
    using func1_1_L11C func1_1_L11D by simp
  from A1 A2 A3 have I:
    f:X→Y  ∀x∈A. b(x)∈B  X∩A = 0  h = f ∪ {⟨x,b(x)⟩. x∈A}
    by auto
  from prems have
    II: h : X∪A → Y∪B  a∉ X∪A  g = h ∪ {⟨a,c⟩}
    using func1_1_L11C by auto
  then have III: ∀x∈X∪A. g(x) = h(x) by (rule func1_1_L11D)
  moreover from I have  ∀x∈X. h(x) = f(x)
    by (rule func1_1_L11C)
  ultimately show ∀x∈X. g(x) = f(x) by simp
  from I have ∀x∈A. h(x) = b(x) by (rule func1_1_L11C)
  with III show ∀x∈A. g(x) = b(x) by simp
```

```
  from II show g(a) = c by (rule func1_1_L11D)
qed
```

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

```
lemma  func1_1_L12:
  assumes A1: B⊆Pow(Y) and A2: B≠0 and A3: f:X→Y
  shows f-(⋂B) = (⋂U∈B. f-(U))
proof
  from A2 show  f-(⋂B) ⊆ (⋂U∈B. f-(U)) by blast
  show (⋂U∈B. f-(U)) ⊆ f-(⋂B)
  proof
    fix x assume A4: x ∈ (⋂U∈B. f-(U))
    from A3 have ∀U∈B. f-(U) ⊆ X using func1_1_L6A by simp
    with A4 have ∀U∈B. x∈X by auto
    with A2 have x∈X by auto
    with A3 have ∃!y. <x,y> ∈ f using Pi_iff_old by simp
    with A2 A4 show x ∈ f-(⋂B) using vimage_iff by blast
  qed
qed
```

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)≠0 shows A≠0
proof (rule ccontr)
  assume A2:¬ A ≠ 0 from A2 A1 show False by simp
qed
```

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)≠0 shows A≠0
proof (rule ccontr)
  assume A2:¬ A ≠ 0 from A2 A1 show False by simp
qed
```

What is the inverse image of a singleton?

```
lemma func1_1_L14: assumes f∈X→Y
  shows f-({y}) = {x∈X. f(x) = y}
  using prems func1_1_L6A vimage_singleton_iff apply_iff by auto
```

A more familiar definition of inverse image.

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

qed

A more familiar definition of image.

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

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

```
lemma image_of_Inter: assumes  A1: f:X→Y and
  A2: I≠0 and A3: ∀i∈I. P(i) ⊆ X
  shows f(⋂i∈I. P(i)) ⊆ ( ⋂i∈I. f(P(i)) )
proof
  fix y assume A4: y ∈ f(⋂i∈I. P(i))
  from A1 A2 A3 have f(⋂i∈I. P(i)) = {f(x). x ∈ ( ⋂i∈I. P(i) )}
    using ZF1_1_L7 func_imagedef by simp
  with A4 obtain x where x ∈ ( ⋂i∈I. P(i) ) and y = f(x)
    by auto
  with A1 A2 A3 show y ∈ ( ⋂i∈I. f(P(i)) ) using func_imagedef
    by auto
qed
```

The image of a nonempty subset of domain is nonempty.

```
lemma func1_1_L15A:
  assumes A1: f: X→Y and A2: A⊆X and A3: A≠0
  shows f(A) ≠ 0
proof -
  from A3 obtain x where x∈A by auto
  with A1 A2 have f(x) ∈ f(A)
    using func_imagedef by auto
  then show f(A) ≠ 0 by auto
qed
```

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

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

```
  using prems func_imagedef by simp
```

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

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

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

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

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 -
  from A2 have {b(x). x∈A} ⊆ X by auto
  with A1 show thesis using func_imagedef by auto
qed
```

What are the values of composition of three functions?

```
lemma func1_1_L18: assumes A1: f:A→B  g:B→C  h:C→D
  and A2: x∈A
  shows
  (h O g O f)(x) ∈ D
  (h O g O f)(x) = h(g(f(x)))
proof -
  from A1 have (h O g O f) : A→D
```

```
    using comp_fun by blast
  with A2 show (h O g O f)(x) ∈ D using apply_funtype
    by simp
  from A1 A2 have (h O g O f)(x) = h( (g O f)(x))
    using comp_fun comp_fun_apply by blast
  with A1 A2 show (h O g O f)(x) = h(g(f(x)))
    using comp_fun_apply by simp
qed
```

4.2 Functions restricted to a set

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

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

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

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

Here is the actual criterion.

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

```
    with A1 A2 A3 A4 D1 have y = g(x) using func1_2_L2 by simp
    with A2 D1 show z∈g using apply_Pair by simp
  qed
qed
```

Which function space a restricted function belongs to?

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

4.3 Constant functions

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

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

Constant function belongs to the function space.

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

Constant function is equal to the constant on its domain.

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

4.4 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`.

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

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

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

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

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

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

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

```
lemma inj_image_of_Inter: assumes A1: f ∈ inj(A,B) and
  A2: I≠0 and A3: ∀i∈I. P(i) ⊆ A
  shows f(⋂i∈I. P(i)) = ( ⋂i∈I. f(P(i)) )
```

```
proof
  from A1 A2 A3 show f(⋂i∈I. P(i)) ⊆ ( ⋂i∈I. f(P(i)) )
    using inj_is_fun image_of_Inter by auto
  from A1 A2 A3 have f:A→B  and ( ⋂i∈I. P(i) ) ⊆ A
    using inj_is_fun ZF1_1_L7 by auto
  then have I: f(⋂i∈I. P(i)) = { f(x). x ∈ ( ⋂i∈I. P(i) ) }
    using func_imagedef by simp
  { fix y assume A4: y ∈ ( ⋂i∈I. f(P(i)) )
    let x = converse(f)(y)
    from A2 obtain i0 where i0 ∈ I by auto
    with A1 A4 have II: y ∈ range(f) using inj_is_fun func1_1_L6
      by auto
    with A1 have III: f(x) = y using right_inverse by simp
    from A1 II have IV: x ∈ A using inj_converse_fun apply_funtype
      by blast
    { fix i assume i∈I
      with A3 A4 III have P(i) ⊆ A and f(x) ∈  f(P(i))
        by auto
      with A1 IV have x ∈ P(i) using inj_point_of_image
        by blast
    } then have ∀i∈I. x ∈ P(i) by simp
    with A2 I have f(x) ∈ f( ⋂i∈I. P(i) )
      by auto
    with III have y ∈  f( ⋂i∈I. P(i) ) by simp
  } then show ( ⋂i∈I. f(P(i)) ) ⊆  f( ⋂i∈I. P(i) )
    by auto
qed
```

This concludes func1.thy.

end

5 Order_ZF.thy

theory Order_ZF **imports** Fol1

begin

This theory file considers various notion related to order. We redefine the notions of a 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 that finite sets are bounded in Finite_ZF.thy.

5.1 Definitions

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

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. The sets that are bounded below and above are also defined, as are bounded sets. Empty sets are defined as bounded. The notation for the definition of an interval may be mysterious for some readers, see Order_ZF_2_L1 for more intuitive notation. We aslo define the maximum (the greater of) two elemnts and the minmum (the smaller of) two elements. We say that a set has a maximum (minimum) if it has an element that is not smaller (not greater, resp.) that any other one. We show that under some conditions this element of the set is unique (if exists). The element with this property is called the maximum (minimum) of the set. The supremum of a set A is defined as the minimum of the set of upper bounds, i.e. the set $\{u.\forall_{a\in A}\langle a,u\rangle \in r\} = \bigcap_{a\in A} r\{a\}$. Infimum is defined analogously. Recall that r-(A)=$\{x : \langle x,y\rangle \in r$ for some $y \in A$ is the inverse image of the set A by relation r. We define a (order) relation to be complete if every nonempty bounded above set has a supremum. This terminolgy may conflict with the one for complete metric space. We will worry about that when we actually define a complete metric space.

constdefs

```
IsTotal (infixl {is total on} 65)
r {is total on} X ≡ (∀a∈X.∀b∈X. <a,b> ∈ r ∨ <b,a> ∈ r)

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

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

IsBoundedAbove(A,r) ≡ ( A=0 ∨ (∃u. ∀x∈A. <x,u> ∈ r))
```

```
IsBoundedBelow(A,r) ≡ (A=0 ∨ (∃l. ∀x∈A. <l,x> ∈ r))

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

Interval(r,a,b) ≡ r{a} ∩ r-{b}

GreaterOf(r,a,b) ≡ (if <a,b> ∈ r then b else a)

SmallerOf(r,a,b) ≡ (if <a,b> ∈ r then a else b)

HasAmaximum(r,A) ≡ ∃M∈A.∀x∈A. <x,M> ∈ r

HasAminimum(r,A) ≡ ∃m∈A.∀x∈A. <m,x> ∈ r

Maximum(r,A) ≡ THE M. M∈A ∧ (∀x∈A. <x,M> ∈ r)

Minimum(r,A) ≡ THE m. m∈A ∧ (∀x∈A. <m,x> ∈ r)

Supremum(r,A) ≡ Minimum(r,⋂a∈A. r{a})

Infimum(r,A) ≡ Maximum(r,⋂a∈A. r-{a})
```

```
IsComplete (_ {is complete})
r {is complete} ≡
∀A. IsBoundedAbove(A,r) ∧ A≠0 ⟶ HasAminimum(r,⋂a∈A. r{a})
```

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

```
lemma Order_ZF_1_L1: assumes r {is total on} X and a∈X
  shows <a,a> ∈ r using prems IsTotal_def by auto
```

A total relation is reflexive.

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

A linear order is partial order.

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

Partial order that is total is linear.

```
lemma Order_ZF_1_L3:
  assumes IsPartOrder(X,r) and r {is total on} X
  shows IsLinOrder(X,r)
```

```
  using prems IsPartOrder_def IsLinOrder_def
  by simp
```

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

```
lemma Order_ZF_1_L4: assumes r {is total on} X and A⊆X
  shows r {is total on} A
  using prems IsTotal_def by auto
```

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

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

5.2 Intervals

In this section we discuss intervals.

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

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

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

```
lemma Order_ZF_2_L1A: assumes x ∈ Interval(r,a,b)
  shows <a,x> ∈ r  <x,b> ∈ r
  using prems  Order_ZF_2_L1 by auto
```

`Order_ZF_2_L1`, implication from right to left.

```
lemma Order_ZF_2_L1B: assumes <a,x> ∈ r  <x,b> ∈ r
  shows x ∈ Interval(r,a,b)
  using prems Order_ZF_2_L1 by simp
```

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

```
lemma Order_ZF_2_L2: assumes refl(X,r)
  and a∈X  b∈X and <a,b> ∈ r
  shows
  a ∈ Interval(r,a,b)
  b ∈ Interval(r,a,b)
  using prems refl_def Order_ZF_2_L1 by auto
```

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

```
lemma Order_ZF_2_L2A: assumes refl(X,r)
  and a∈X  b∈X and <a,b> ∈ r
```

```
  shows Interval(r,a,b) ≠ 0
proof -
  from prems have a ∈ Interval(r,a,b)
    using Order_ZF_2_L2 by simp
  then show Interval(r,a,b) ≠ 0 by auto
qed
```

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

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

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

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

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

```
lemma Order_ZF_2_L5: assumes A1: trans(r) and A2: <a,b> ∉ r
  shows Interval(r,a,b) = 0
proof (rule ccontr)
  assume Interval(r,a,b)≠0 then obtain x where x ∈ Interval(r,a,b)
    by auto
  with A1 A2 show False using Order_ZF_2_L1A Fol1_L3 by fast
qed
```

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

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

5.3 Bounded sets

In this section we consider properties of bounded sets.

For reflexive relations singletons are bounded.

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

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

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

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

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

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

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

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

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

```
  proof (cases x∈A)
    assume x∈A
    with A4 T1 have <x,u> ∈ r ∧ <u,v> ∈ r by simp
    with A2 show ⟨x,v⟩ ∈ r by (rule Fol1_L3)
  next assume x∉A
    with A5 A4 T2 have <x,w> ∈ r ∧ <w,v> ∈ r by simp
    with A2 show ⟨x,v⟩ ∈ r by (rule Fol1_L3)
  qed
qed
```

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

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

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

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

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

```
lemma Order_ZF_3_L5B:
```

```
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: l∈X m∈X
  and A4: ∀x∈A. <l,x> ∈ r ∀x∈B. <m,x> ∈ r
  shows ∀x∈A∪B. ⟨SmallerOf(r,l,m),x⟩ ∈ r
proof
  let k = SmallerOf(r,l,m)
  from A1 A3 have T1: <k,l> ∈ r and T2: <k,m> ∈ r
    using Order_ZF_3_L2 by auto
  fix x assume A5: x∈A∪B show ⟨k,x⟩ ∈ r
  proof (cases x∈A)
    assume x∈A
    with A4 T1 have <k,l> ∈ r ∧ <l,x> ∈ r by simp
    with A2 show ⟨k,x⟩ ∈ r by (rule Fol1_L3)
  next assume x∉A
    with A5 A4 T2 have <k,m> ∈ r ∧ <m,x> ∈ r by simp
    with A2 show ⟨k,x⟩ ∈ r by (rule Fol1_L3)
  qed
qed
```

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

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

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

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

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

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

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

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

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

A sufficient condition for a set to be bounded below.

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

A sufficient condition for a set to be bounded above.

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

Intervals are bounded.

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

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

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

```
lemma Order_ZF_3_L12: assumes IsBoundedBelow(A,r) and B⊆A
  shows IsBoundedBelow(B,r)
  using prems IsBoundedBelow_def by auto
```

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

```
lemma Order_ZF_3_L13: assumes IsBoundedAbove(A,r) and B⊆A
  shows IsBoundedAbove(B,r)
  using prems IsBoundedAbove_def by auto
```

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.

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

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

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

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

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

5.4 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.thy) that every finite set has well-defined maximum and minimum.

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

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

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

```
lemma Order_ZF_4_L2: assumes A1: antisym(r) and A2: HasAminimum(r,A)
  shows ∃!m. m∈A ∧ (∀x∈A. <m,x> ∈ r)
proof
```

```
  from A2 show ∃m. m ∈ A ∧ (∀x∈A. ⟨m, x⟩ ∈ r)
    using HasAminimum_def by auto
  fix m1 m2 assume
    A2: m1 ∈ A ∧ (∀x∈A. ⟨m1, x⟩ ∈ r) m2 ∈ A ∧ (∀x∈A. ⟨m2, x⟩ ∈ r)
    then have ⟨m1,m2⟩ ∈ r ⟨m2,m1⟩ ∈ r by auto
    with A1 show m1=m2 by (rule Fol1_L4)
qed
```

Maximum of a set has desired properties.

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

Minimum of a set has desired properties.

```
lemma Order_ZF_4_L4: assumes A1: antisym(r) and A2: HasAminimum(r,A)
  shows Minimum(r,A) ∈ A ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
proof -
  let Min = THE m. m∈A ∧ (∀x∈A. <m,x> ∈ r)
  from A1 A2 have ∃!m. m∈A ∧ (∀x∈A. <m,x> ∈ r)
    by (rule Order_ZF_4_L2)
  then have Min ∈ A ∧ (∀x∈A. <Min,x> ∈ r)
    by (rule theI)
  then show Minimum(r,A) ∈ A ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
    using Minimum_def by auto
qed
```

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

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

```
  moreover from T1 have L ∈ A∪B using GreaterOf_def IsTotal_def
    by simp
  ultimately show HasAmaximum(r,A∪B) using HasAmaximum_def by auto
qed
```

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

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

Set that has a maximum is bounded above.

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

Set that has a minimum is bounded below.

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

For reflexive relations singletons have a minimum and maximum.

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

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

```
lemma Order_ZF_4_L9:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: A⊆X and A4: a∈X and A5: HasAmaximum(r,A)
  shows HasAmaximum(r,A∪{a})
```

```
proof -
  from A3 A4 have A∪{a} ⊆ X by auto
  with A1 have r {is total on} (A∪{a})
    using Order_ZF_1_L4 by blast
  moreover from A1 A2 A4 A5 have
    trans(r) HasAmaximum(r,A) by auto
  moreover from A1 A4 have HasAmaximum(r,{a})
    using total_is_refl Order_ZF_4_L8 by blast
  ultimately show HasAmaximum(r,A∪{a}) by (rule Order_ZF_4_L5)
qed
```

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

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

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

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

```
      using Order_ZF_3_L12 by simp
    moreover have IsBoundedAbove(L,r)
      by (rule Order_ZF_3_L15)
    ultimately have IsBoundedAbove(L,r) ∧ IsBoundedBelow(L,r)
      by blast
    then show IsBounded(L,r) using IsBounded_def
      by simp
  qed
  ultimately have IsBounded(L,r) ∧ L ≠ 0 by blast
  with A4 have HasAminimum(r,L) by simp
  then obtain m where I: m∈L and II: ∀x∈L. <m,x> ∈ r
    using HasAminimum_def by auto
  then have III: ⟨m,b⟩ ∈ r by simp
  from I have m∈B by simp
  moreover have ∀x∈B. ⟨m,x⟩ ∈ r
  proof
    fix x assume A7: x∈B
    from A3 A6 have B⊆X using Order_ZF_3_L1B by blast
    with A1 A7 T1 have x ∈  L ∪ {x∈B. ⟨b,x⟩ ∈ r}
      using Order_ZF_1_L5 by simp
    then have x∈L ∨ ⟨b,x⟩ ∈ r by auto
    moreover
    { assume x∈L
      with II have ⟨m,x⟩ ∈ r by simp }
    moreover
    { assume ⟨b,x⟩ ∈ r
      with A2 III have trans(r) and ⟨m,b⟩ ∈ r ∧ ⟨b,x⟩ ∈ r
        by auto
      then have  ⟨m,x⟩ ∈ r by (rule Fol1_L3) }
    ultimately show ⟨m,x⟩ ∈ r by auto
  qed
  ultimately show HasAminimum(r,B) using HasAminimum_def
    by auto
qed
```

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

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

```
  with A1 T have T2: b ∈ U
    using total_is_refl refl_def by simp
  then have U ≠ 0 by auto
  moreover have IsBounded(U,r)
  proof -
    have U ⊆ B by auto
    with A6 have IsBoundedAbove(U,r)
      using Order_ZF_3_L13 by blast
    moreover have IsBoundedBelow(U,r)
      using IsBoundedBelow_def by auto
    ultimately have IsBoundedAbove(U,r) ∧ IsBoundedBelow(U,r)
      by blast
    then show IsBounded(U,r) using IsBounded_def
      by simp
  qed
  ultimately have IsBounded(U,r) ∧ U ≠ 0 by blast
  with A4 have HasAmaximum(r,U) by simp
  then obtain m where I: m∈U and II: ∀x∈U. ⟨x,m⟩ ∈ r
    using HasAmaximum_def by auto
  then have III: ⟨b,m⟩ ∈ r by simp
  from I have m∈B by simp
  moreover have ∀x∈B. ⟨x,m⟩ ∈ r
  proof
    fix x assume A7: x∈B
    from A3 A6 have B⊆X using Order_ZF_3_L1A by blast
    with A1 A7 T1 have x ∈ {x∈B. ⟨x,b⟩ ∈ r} ∪ U
      using Order_ZF_1_L5 by simp
    then have x∈U ∨ ⟨x,b⟩ ∈ r by auto
    moreover
    { assume x∈U
      with II have ⟨x,m⟩ ∈ r by simp }
    moreover
    { assume ⟨x,b⟩ ∈ r
      with A2 III have trans(r) and ⟨x,b⟩ ∈ r ∧ ⟨b,m⟩ ∈ r
        by auto
      then have  ⟨x,m⟩ ∈ r by (rule Fol1_L3) }
    ultimately show ⟨x,m⟩ ∈ r by auto
  qed
  ultimately show HasAmaximum(r,B) using HasAmaximum_def
    by auto
qed
```

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

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

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

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

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

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

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

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

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

```
lemma Order_ZF_4_L16:
  assumes A1: antisym(r) and A2: r {is total on} X and
  A3: A⊆X and
```

```
  A4: ¬HasAmaximum(r,A) and
  A5: x∈A
  shows ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x
proof -
  { assume A6: ∀y∈A. ⟨x,y⟩ ∉ r ∨ y=x
    have ∀y∈A. ⟨y,x⟩ ∈ r
    proof
      fix y assume A7: y∈A
      with A6 have ⟨x,y⟩ ∉ r ∨ y=x by simp
      with A2 A3 A5 A7 show ⟨y,x⟩ ∈ r
        using IsTotal_def Order_ZF_1_L1 by auto
    qed
    with A5 have ∃x∈A.∀y∈A. ⟨y,x⟩ ∈ r
      by auto
    with A4 have False using HasAmaximum_def by simp
  } then show ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x by auto
qed
```

5.5 Supremum and Infimum

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

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

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

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

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

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

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

```
    using Order_ZF_4_L4 Supremum_def by simp
qed
```

Infimum is greater or equal than any lower bound.

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

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

```
lemma Order_ZF_5_L5: assumes A1: antisym(r) and A2: A≠0 and
  A3: ∀x∈A. ⟨x,z⟩ ∈ r and
  A4: ∀y. (∀x∈A. ⟨x,y⟩ ∈ r) ⟶ ⟨z,y⟩ ∈ r
  shows
  HasAminimum(r,⋂a∈A. r{a})
  z = Supremum(r,A)
proof -
  let B = ⋂a∈A. r{a}
  from A2 A3 A4 have I: z ∈ B   ∀y∈B. ⟨z,y⟩ ∈ r
    by auto
  then show HasAminimum(r,⋂a∈A. r{a})
    using HasAminimum_def by auto
  from A1 I show z = Supremum(r,A)
    using Order_ZF_4_L15 Supremum_def by simp
qed
```

If a set has a maximum, then the maximum is the supremum.

```
lemma Order_ZF_5_L6:
  assumes A1:  antisym(r) and A2: A≠0 and
  A3: HasAmaximum(r,A)
  shows
  HasAminimum(r,⋂a∈A. r{a})
  Maximum(r,A) = Supremum(r,A)
proof -
  let M = Maximum(r,A)
  from A1 A3 have I: M ∈ A and II: ∀x∈A. ⟨x,M⟩ ∈ r
    using Order_ZF_4_L3 by auto
  from I have III: ∀y. (∀x∈A. ⟨x,y⟩ ∈ r) ⟶ ⟨M,y⟩ ∈ r
    by simp
  with A1 A2 II show HasAminimum(r,⋂a∈A. r{a})
```

```
    by (rule Order_ZF_5_L5)
  from A1 A2 II III show M = Supremum(r,A)
    by (rule Order_ZF_5_L5)
qed
```

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⊆X  A≠0 and A5: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r
  shows
  Supremum(r,A) ∈ X
  ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r
proof -
  from A5 have IsBoundedAbove(A,r) using IsBoundedAbove_def
    by auto
  with A3 A4 have HasAminimum(r,⋂a∈A. r{a})
    using IsComplete_def by simp
  with A2 have Minimum(r,⋂a∈A. r{a}) ∈ ( ⋂a∈A. r{a} )
    using Order_ZF_4_L4 by simp
  moreover have Minimum(r,⋂a∈A. r{a}) = Supremum(r,A)
    using Supremum_def by simp
  ultimately have I: Supremum(r,A) ∈  ( ⋂a∈A. r{a} )
    by simp
  moreover from A4 obtain a where a∈A by auto
  ultimately have ⟨a,Supremum(r,A)⟩ ∈ r using Order_ZF_5_L1
    by simp
  with A1 show Supremum(r,A) ∈ X by auto
  from I show ∀x∈A. ⟨x,Supremum(r,A)⟩ ∈ r using Order_ZF_5_L1
    by simp
qed
```

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

```
lemma Order_ZF_5_L8:
  assumes A1: r ⊆ X×X  and A2: IsLinOrder(X,r) and
  A3: r {is complete} and
  A4: A⊆X  A≠0 and A5: ∃x∈X. ∀y∈A. ⟨y,x⟩ ∈ r and
  A6: ⟨y,Supremum(r,A)⟩ ∈ r   y ≠ Supremum(r,A)
  shows ∃z∈A. ⟨y,z⟩ ∈ r ∧ y ≠ z
proof -
  from A2 have
    I: antisym(r) and
    II: trans(r) and
    III: r {is total on} X
    using IsLinOrder_def by auto
  from A1 A6 have T1: y∈X by auto
  { assume A7: ∀z ∈ A. ⟨y,z⟩ ∉ r ∨ y=z
    from A4 I have antisym(r) and A≠0 by auto
```

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

5.6 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.

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

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

```
lemma def_of_strict_ver: shows
  ⟨x,y⟩ ∈ StrictVersion(r) ⟷ ⟨x,y⟩ ∈ r ∧ x≠y
  using StrictVersion_def domain_def by auto
```

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

```
lemma strict_of_antisym:
```

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

The strict version of totality.

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

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

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

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

```
corollary strict_lin_trich: assumes A1: IsLinOrder(X,r) and
  A2: a∈X  b∈X and
  A3: s = StrictVersion(r)
  shows Exactly_1_of_3_holds(⟨a,b⟩ ∈ s, a=b,⟨b,a⟩ ∈ s)
  using prems IsLinOrder_def strict_ans_tot_trich by auto
```

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

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

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

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

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

```
lemma strict_of_transB:
  assumes A1: trans(r) and A2: antisym(r)
  shows trans(StrictVersion(r))
proof -
  let s = StrictVersion(r)
  from A1 A2 have
    ∀ x y z. ⟨x, y⟩ ∈ s ∧ ⟨y, z⟩ ∈ s ⟶ ⟨x, z⟩ ∈ s
    using strict_of_transA by blast
  then show trans(StrictVersion(r)) by (rule Fol1_L2)
qed
```

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

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

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

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


end
```

6 func_ZF.thy

theory `func_ZF` **imports** `Order func1 Order_ZF`

begin

In this theory we consider properties of functions that are binary operations, that is they map $X \times X$ into X. We also consider some properties of functions related to order.

6.1 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.

```
constdefs
  Lift2FcnSpce (infix {lifted to function space over} 65)
  f {lifted to function space over} X ≡
  {<p,g> ∈ ((X→range(f))×(X→range(f)))×(X→range(f)).
  {<x,y> ∈ X×range(f). f<fst(p)(x),snd(p)(x)> = y} = g}
```

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,y> ∈ X×range(f). f<fst(p)(x),snd(p)(x)> = y} : X→range(f)
  proof -
    have ∀x∈X. f<fst(p)(x),snd(p)(x)> ∈ range(f)
    proof
      fix x assume A3:x∈X
      let p = <fst(p)(x),snd(p)(x)>
      from A2 A3 have
        fst(p)(x) ∈ range(f)  snd(p)(x) ∈ range(f)
        using apply_type by auto
      with A1 have p ∈ Y×Y
        using func1_1_L5B by blast
      with A1 have <p, f(p)> ∈ f
        using apply_Pair by simp
      with A1 show
        f(p) ∈ range(f)
        using rangeI by simp
```

```
    qed
    then show thesis using func1_1_L11A by simp
qed
```

The values of the lift are defined by the value of the liftee in a natural way.

```
lemma func_ZF_1_L2:
  assumes f : Y×Y→Y
  and p∈(X→range(f))×(X→range(f)) and x∈X
  and P = {<x,y> ∈ X×range(f). f<fst(p)(x),snd(p)(x)> = y}
  shows P(x) = f⟨fst(p)(x),snd(p)(x)⟩
  using prems func_ZF_1_L1 func1_1_L11B by simp
```

Function lifted to a function space results in a function space operator.

```
lemma func_ZF_1_L3:
  assumes f ∈ Y×Y→Y
  and F = f {lifted to function space over} X
  shows F : (X→range(f))×(X→range(f))→(X→range(f))
  using prems Lift2FcnSpce_def func_ZF_1_L1 func1_1_L11A by simp
```

The values of the lift are defined by the values of the liftee in the natural way. For some reason we need to be extremely detailed and explicit to be able to apply func1_3_L2. simp and auto fail miserably here.

```
lemma func_ZF_1_L4:
  assumes A1: f : Y×Y→Y
  and A2: F = f {lifted to function space over} X
  and A3: s:X→range(f) r:X→range(f)
  and A4: x∈X
  shows (F<s,r>)(x) = f<s(x),r(x)>
proof -
  let P = {<x,y> ∈ X×range(f). f<s(x),r(x)> = y}
  let p = <s,r>
  from A1 have f ∈ Y×Y→Y .
  moreover from A3 have
    p ∈ (X→range(f))×(X→range(f))
    by simp
  moreover from A4 have x∈X .
  moreover have
    P = {<x,y> ∈ X×range(f). f<fst(p)(x),snd(p)(x)> = y}
    by simp
  ultimately have P(x) = f⟨fst(p)(x),snd(p)(x)⟩
    by (rule func_ZF_1_L2)
  with A1 A2 A3 show thesis using func_ZF_1_L3 Lift2FcnSpce_def func1_1_L11B
    by simp
qed
```

6.2 Associative and commutative operations

In this section we define associative and commutative oparations and prove that they remain such when we lift them to a function space.

```
constdefs

  IsAssociative (infix {is associative on} 65)
  f {is associative on} G ≡ f ∈ G×G→G ∧
  (∀ x ∈ G. ∀ y ∈ G. ∀ z ∈ G.
  ( f(<f(<x,y>),z>) = f( < x,f(<y,z>)> )))

  IsCommutative (infix {is commutative on} 65)
  f {is commutative on} G ≡ ∀x∈G. ∀y∈G. f<x,y> = f<y,x>
```

The lift of a commutative function is commutative.

```
lemma func_ZF_2_L1:
  assumes A1: f : G×G→G
  and A2: F = f {lifted to function space over} X
  and A3: s : X→range(f) r : X→range(f)
  and A4: f {is commutative on} G
  shows F<s,r> = F<r,s>
proof -
  from A1 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using func_ZF_1_L3 by simp
  with A3 have
    F<s,r> : X→range(f) F<r,s> : X→range(f)
    using apply_type by auto
  moreover have
    ∀x∈X. (F<s,r>)(x) = (F<r,s>)(x)
  proof
    fix x assume A5:x∈X
    from A1 have range(f)⊆G
      using func1_1_L5B by simp
    with A3 A5 have T1:s(x) ∈ G r(x) ∈ G
      using apply_type by auto
    with A1 A2 A3 A4 A5 show
      (F<s,r>)(x) = (F<r,s>)(x)
      using func_ZF_1_L4 IsCommutative_def by simp
  qed
  ultimately show thesis using fun_extension_iff
    by simp
qed
```

The lift of a commutative function is commutative on the function space.

```
lemma func_ZF_2_L2:
  assumes f : G×G→G
  and f {is commutative on} G
  and F = f {lifted to function space over} X
  shows F {is commutative on} (X→range(f))
  using prems IsCommutative_def func_ZF_2_L1 by simp
```

The lift of an associative function is associative.

```
lemma func_ZF_2_L3:
  assumes A2: F = f {lifted to function space over} X
  and A3: s : X→range(f) r : X→range(f) q : X→range(f)
  and A4: f {is associative on} G
  shows F⟨F<s,r>,q⟩ = F⟨s,F<r,q>⟩
proof -
  from A4 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using IsAssociative_def func_ZF_1_L3 by auto
  with A3 have T1:
    F<s,r> : X→range(f)
    F<r,q> : X→range(f)
    F<F<s,r>,q> : X→range(f)
    F<s,F<r,q> >: X→range(f)
    using apply_type by auto
  moreover have
    ∀x∈X. (F⟨F<s,r>,q⟩)(x) = (F⟨s,F<r,q>⟩)(x)
  proof
    fix x assume A5:x∈X
    from A4 have T2:f:G×G→G
      using IsAssociative_def by simp
    then have range(f)⊆G
      using func1_1_L5B by simp
    with A3 A5 have
      s(x) ∈ G r(x) ∈ G q(x) ∈ G
      using apply_type by auto
    with T2 A2 T1 A3 A5 A4 show
      (F⟨F<s,r>,q⟩)(x) = (F⟨s,F<r,q>⟩)(x)
      using func_ZF_1_L4 IsAssociative_def by simp
  qed
  ultimately show thesis using fun_extension_iff
    by simp
qed
```

The lift of an associative function is associative on the function space.

```
lemma func_ZF_2_L4:
  assumes A1: f {is associative on} G
  and A2: F = f {lifted to function space over} X
  shows F {is associative on} (X→range(f))
proof -
  from A1 A2 have
    F : (X→range(f))×(X→range(f))→(X→range(f))
    using IsAssociative_def func_ZF_1_L3 by auto
  moreover from A1 A2 have
    ∀s ∈ X→range(f). ∀ r ∈ X→range(f). ∀q ∈ X→range(f).
    F<F<s,r>,q> = F<s,F<r,q> >
    using func_ZF_2_L3 by simp
  ultimately show thesis using IsAssociative_def
    by simp
```

qed

6.3 Restricting operations

In this section we consider when restriction of the operation to a set inherits properties like commutativity and associativity.

The commutativity is inherited when restricting a function to a set.

```
lemma func_ZF_4_L1:
  assumes A1: f:X×X→Y and A2: A⊆X
  and A3: f {is commutative on} X
  shows restrict(f,A×A) {is commutative on} A
proof -
  { fix x y assume A4: x∈A ∧ y∈A
    with A2 A3 have
      f<x,y> = f<y,x>
      using IsCommutative_def by auto
    moreover from A4 have
      restrict(f,A×A)<x,y> = f<x,y>
      restrict(f,A×A)<y,x> = f<y,x>
      using restrict_if by auto
    ultimately have
      restrict(f,A×A)<x,y> = restrict(f,A×A)<y,x>
      by simp }
  then show thesis using IsCommutative_def by simp
qed
```

Next we define sets closed with respect to an operation.

```
constdefs
  IsOpClosed (infix {is closed under} 65)
  A {is closed under} f ≡ ∀x∈A. ∀y∈A. f<x,y> ∈ A
```

Associative operation restricted to a set that is closed with resp. to this operation is associative.

```
lemma func_ZF_4_L2:assumes A1: f {is associative on} X
  and A2: A⊆X and A3: A {is closed under} f
  and A4: x∈A y∈A z∈A
  and A5: g = restrict(f,A×A)
  shows g⟨g<x,y>,z⟩ = g⟨x,g<y,z>⟩
proof -
  from A4 A2 have T1:
    x∈X y∈X z∈X
    by auto
  from A3 A4 A5 have
    g<g<x,y>,z> = f<f<x,y>,z>
    g<x,g<y,z> > = f<x,f<y,z> >
    using IsOpClosed_def restrict_if by auto
  moreover from A1 T1 have
```

```
    f<f<x,y>,z> = f<x,f<y,z> >
    using IsAssociative_def by simp
  ultimately show thesis by simp
qed
```

Associative operation restricted to a set that is closed with resp. to this operation is associative on the set.

```
lemma func_ZF_4_L3: assumes A1: f {is associative on} X
  and A2: A⊆X and A3: A {is closed under} f
  shows restrict(f,A×A) {is associative on} A
proof -
  let g = restrict(f,A×A)
  from A1 have f:X×X→X
    using IsAssociative_def by simp
  moreover from A2 have A×A ⊆ X×X by auto
  moreover from A3 have ∀p ∈ A×A. g(p) ∈ A
    using IsOpClosed_def restrict_if by auto
  ultimately have g : A×A→A
    using func1_2_L4 by simp
  moreover from  A1 A2 A3 have
    ∀ x ∈ A. ∀ y ∈ A. ∀ z ∈ A.
    g<g<x,y>,z> = g< x,g<y,z> >
    using func_ZF_4_L2 by simp
  ultimately show thesis
    using IsAssociative_def by simp
qed
```

The essential condition to show that if a set A is closed with respect to an operation, then it is closed under this operation restricted to any superset of A.

```
lemma func_ZF_4_L4: assumes A {is closed under} f
  and A⊆B and x∈A  y∈A and g = restrict(f,B×B)
  shows g<x,y> ∈ A
  using prems IsOpClosed_def restrict by auto
```

If a set A is closed under an operation, then it is closed under this operation restricted to any superset of A.

```
lemma func_ZF_4_L5:
  assumes A1: A {is closed under} f
  and A2: A⊆B
  shows A {is closed under} restrict(f,B×B)
proof -
  let g = restrict(f,B×B)
  from A1 A2 have ∀x∈A. ∀y∈A. g<x,y> ∈ A
    using func_ZF_4_L4 by simp
  then show thesis using IsOpClosed_def by simp
qed
```

The essential condition to show that intersection of sets that are closed with respect to an operation is closed with respect to the operation.

```
lemma func_ZF_4_L6:
  assumes A {is closed under} f
  and B {is closed under} f
  and x ∈ A∩B y∈ A∩B
  shows f<x,y> ∈ A∩B using prems IsOpClosed_def by auto
```

Intersection of sets that are closed with respect to an operation is closed under the operation.

```
lemma func_ZF_4_L7:
  assumes A {is closed under} f
  B {is closed under} f
  shows A∩B {is closed under} f
  using prems IsOpClosed_def by simp
```

6.4 Composition

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. In this section we consider the corresponding two-argument ZF-function (binary operation), that is a subset of $((X \to X) \times (X \to X)) \times (X \to X)$.

```
constdefs
  Composition(X) ≡
  {<p,f> ∈ ((X→X)×(X→X))×(X→X). fst(p) O snd(p) = f}
```

Composition operation is a function that maps $(X \to X) \times (X \to X)$ into $X \to X$.

```
lemma func_ZF_5_L1: shows Composition(X) : (X→X)×(X→X)→(X→X)
  using comp_fun Composition_def func1_1_L11A by simp
```

The value of the composition operation is the composition of arguments.

```
lemma func_ZF_5_L2: assumes f:X→X g:X→X
  shows Composition(X)<f,g> = f O g
  using prems func_ZF_5_L1 Composition_def func1_1_L11B by simp
```

What is the falue of a composition on an argument?

```
lemma func_ZF_5_L3: assumes f:X→X and g:X→X and x∈X
  shows (Composition(X)<f,g>)(x) = f(g(x))
  using prems func_ZF_5_L2 comp_fun_apply by simp
```

The essential condition to show that composition is associative.

```
lemma func_ZF_5_L4: assumes A1: f:X→X g:X→X h:X→X
  and A2: C = Composition(X)
  shows C⟨C<f,g>,h⟩ = C⟨ f,C<g,h>⟩
```

```
proof -
  from A2 have C : ((X→X)×(X→X))→(X→X)
    using func_ZF_5_L1 by simp
  with A1 have T1:
    C<f,g> : X→X
    C<g,h> : X→X
    C<C<f,g>,h> : X→X
    C< f,C<g,h> > : X→X
    using apply_funtype by auto
  moreover have
    ∀ x ∈ X. C⟨C<f,g>,h⟩(x) = C⟨f,C<g,h>⟩(x)
  proof
    fix x assume A3:x∈X
    with A1 A2 T1 have
      C<C<f,g>,h>  (x) = f(g(h(x)))
      C< f,C<g,h> >(x) = f(g(h(x)))
      using func_ZF_5_L3 apply_funtype by auto
    then show C⟨C<f,g>,h⟩(x) = C⟨ f,C<g,h>⟩(x)
      by simp
    qed
  ultimately show thesis using fun_extension_iff by simp
qed
```

Composition is an associative operation on $X \to X$ (the space of functions that map X into itself).

```
lemma func_ZF_5_L5: shows Composition(X) {is associative on} (X→X)
proof -
  let C = Composition(X)
  have ∀f∈X→X. ∀g∈X→X. ∀h∈X→X.
    C<C<f,g>,h> = C< f,C<g,h> >
    using func_ZF_5_L4 by simp
  then show thesis using func_ZF_5_L1 IsAssociative_def
    by simp
qed
```

6.5 Identity function

In this section we show some additional facts about the identity function defined in the standard Isabelle's Perm.thy file.

Composing a function with identity does not change the function.

```
lemma func_ZF_6_L1A: assumes A1: f : X→X
  shows Composition(X)<f,id(X)> = f
  Composition(X)<id(X),f> = f
proof -
  have Composition(X) : (X→X)×(X→X)→(X→X)
    using func_ZF_5_L1 by simp
  with A1 have Composition(X)<id(X),f> : X→X
    Composition(X)<f,id(X)> : X→X
```

```
    using id_type apply_funtype by auto
  moreover from A1 have f : X→X .
  moreover from A1 have
    ∀x∈X. (Composition(X)<id(X),f>)(x) = f(x)
    ∀x∈X. (Composition(X)<f,id(X)>)(x) = f(x)
    using id_type func_ZF_5_L3 apply_funtype id_conv
    by auto
  ultimately show Composition(X)<id(X),f> = f
    Composition(X)<f,id(X)> = f
    using fun_extension_iff by auto
qed
```

6.6 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.thy 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.

```
constdefs
  IsDistributive(X,A,M) ≡ (∀a∈X.∀b∈X.∀c∈X.
  M⟨a,A<b,c>⟩ = A⟨M<a,b>,M<a,c>⟩ ∧
  M⟨A<b,c>,a⟩ = A⟨M<b,a>,M<c,a> ⟩)
```

The essential condition to show that distributivity is preserved by restrictions to sets that are closed with respect to both operations.

```
lemma func_ZF_7_L1:
  assumes A1: IsDistributive(X,A,M)
  and A2: Y⊆X
  and A3: Y {is closed under} A  Y {is closed under} M
  and A4: Ar = restrict(A,Y×Y) Mr = restrict(M,Y×Y)
  and A5: a∈Y  b∈Y  c∈Y
  shows Mr⟨ a,Ar<b,c> ⟩  = Ar⟨ Mr<a,b>,Mr<a,c> ⟩  ∧
  Mr⟨ Ar<b,c>,a ⟩ = Ar⟨ Mr<b,a>,Mr<c,a> ⟩
proof
  from A3 A5 have A<b,c> ∈ Y  M<a,b> ∈ Y  M<a,c> ∈ Y
    M<b,a> ∈ Y  M<c,a> ∈ Y using IsOpClosed_def by auto
  with A5 A4 have T1:Ar<b,c> ∈ Y Mr<a,b> ∈ Y Mr<a,c> ∈ Y
    Mr<b,a> ∈ Y Mr<c,a> ∈ Y
    using restrict by auto
  with A1 A2 A4 A5 show Mr⟨ a,Ar<b,c> ⟩  = Ar⟨ Mr<a,b>,Mr<a,c> ⟩
    Mr⟨ Ar<b,c>,a ⟩ = Ar⟨ Mr<b,a>,Mr<c,a> ⟩
    using restrict IsDistributive_def by auto
```

qed

Distributivity is preserved by restrictions to sets that are closed with respect to both operations.

```
lemma func_ZF_7_L2:
  assumes IsDistributive(X,A,M)
  and Y⊆X
  and Y {is closed under} A
  Y {is closed under} M
  and A_r = restrict(A,Y×Y) M_r = restrict(M,Y×Y)
  shows IsDistributive(Y,A_r,M_r)
proof -
  from prems have ∀a∈Y.∀b∈Y.∀c∈Y.
    M_r⟨ a,A_r<b,c> ⟩ = A_r⟨ M_r<a,b>,M_r<a,c> ⟩ ∧
    M_r⟨ A_r<b,c>,a ⟩ = A_r⟨ M_r<b,a>,M_r<c,a> ⟩
    using func_ZF_7_L1 by simp
  then show thesis using IsDistributive_def by simp
qed
```

6.7 Functions and order

This section deals with functions between ordered sets.

If every value of a function on a set is bounded below by a constant, then the image of the set is bounded below.

```
lemma func_ZF_8_L1:
  assumes f:X→Y and A⊆X and ∀x∈A. ⟨L,f(x)⟩ ∈ r
  shows IsBoundedBelow(f(A),r)
proof -
  from prems have ∀y ∈ f(A). ⟨L,y⟩ ∈ r
    using func_imagedef by simp
  then show IsBoundedBelow(f(A),r)
    by (rule Order_ZF_3_L9)
qed
```

If every value of a function on a set is bounded above by a constant, then the image of the set is bounded above.

```
lemma func_ZF_8_L2:
  assumes f:X→Y and A⊆X and ∀x∈A. ⟨f(x),U⟩ ∈ r
  shows IsBoundedAbove(f(A),r)
proof -
  from prems have ∀y ∈ f(A). ⟨y,U⟩ ∈ r
    using func_imagedef by simp
  then show IsBoundedAbove(f(A),r)
    by (rule Order_ZF_3_L10)
qed
```

6.8 Projections 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)`.

```
constdefs
  SliceProjection(X) ≡ {⟨p,fst(p)⟩. p ∈ X }
```

A slice projection is a bijection between $X \times \{y\}$ and X.

```
lemma slice_proj_bij: shows
  SliceProjection(X×{y}): X×{y} → X
  domain(SliceProjection(X×{y})) = X×{y}
  ∀p∈X×{y}. SliceProjection(X×{y})(p) = fst(p)
  SliceProjection(X×{y}) ∈ bij(X×{y},X)
proof -
  let P = SliceProjection(X×{y})
  have  ∀p ∈ X×{y}. fst(p) ∈ X by simp
  moreover from this have
    {⟨p,fst(p)⟩. p ∈ X×{y} } : X×{y} → X
    by (rule ZF_fun_from_total)
  ultimately show
    I: P: X×{y} → X and II: ∀p∈X×{y}. P(p) = fst(p)
    using ZF_fun_from_tot_val SliceProjection_def by auto
  hence
    ∀a ∈ X×{y}. ∀ b ∈ X×{y}. P(a) = P(b) ⟶ a=b
    by auto
  with I have P ∈ inj(X×{y},X) using inj_def
    by simp
  moreover from II have ∀x∈X. ∃p∈X×{y}. P(p) = x
    by simp
  with I have P ∈ surj(X×{y},X) using surj_def
    by simp
  ultimately show P ∈ bij(X×{y},X)
    using bij_def by simp
  from I show domain(SliceProjection(X×{y})) = X×{y}
    using func1_1_L1 by simp
qed
```

6.9 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.thy` 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)`.

```
constdefs
  InducedRelation(f,R) ≡
  {p ∈ domain(f)×domain(f). ⟨f(fst(p)),f(snd(p))⟩ ∈ R}
```

A reformulation of the definition of the relation induced by a function.

```
lemma def_of_ind_relA:
  assumes ⟨x,y⟩ ∈ InducedRelation(f,R)
  shows ⟨f(x),f(y)⟩ ∈ R
  using prems InducedRelation_def by simp
```

A reformulation of the definition of the relation induced by a function, kind of converse of `def_of_ind_relA`.

```
lemma def_of_ind_relB: assumes f:A→B and
  x∈A  y∈A and ⟨f(x),f(y)⟩ ∈ R
  shows ⟨x,y⟩ ∈ InducedRelation(f,R)
  using prems func1_1_L1 InducedRelation_def by simp
```

A property of order isomorphisms that is missing from standard Isabelle's `Order.thy`.

```
lemma ord_iso_apply_conv:
  assumes f ∈ ord_iso(A,r,B,R) and
  ⟨f(x),f(y)⟩ ∈ R and x∈A  y∈A
  shows ⟨x,y⟩ ∈ r
  using prems ord_iso_def by simp
```

The next lemma tells us where the induced relation is defined

```
lemma ind_rel_domain:
  assumes  R ⊆ B×B and f:A→B
  shows InducedRelation(f,R) ⊆ A×A
  using prems func1_1_L1 InducedRelation_def
  by auto
```

A bijection is an order homomorphisms between a relation and the induced one.

```
lemma bij_is_ord_iso: assumes A1: f ∈ bij(A,B)
  shows f ∈ ord_iso(A,InducedRelation(f,R),B,R)
proof -
  let r = InducedRelation(f,R)
  { fix x y assume A2: x∈A  y∈A
    have ⟨x,y⟩ ∈ r ←→ ⟨f(x),f(y)⟩ ∈ R
    proof
      assume ⟨x,y⟩ ∈ r then show ⟨f(x),f(y)⟩ ∈ R
```

```
      using def_of_ind_relA by simp
    next assume ⟨f(x),f(y)⟩ ∈ R
      with A1 A2 show ⟨x,y⟩ ∈ r
        using bij_is_fun def_of_ind_relB by blast
    qed }
  with A1 show f ∈ ord_iso(A,InducedRelation(f,R),B,R)
    using ord_isoI by simp
qed
```

An order isomoprhism preserves antisymmetry.

```
lemma ord_iso_pres_antsym: assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: antisym(R)
  shows antisym(r)
proof -
  { fix x y
    assume A4: ⟨x,y⟩ ∈ r   ⟨y,x⟩ ∈ r
    from A1 have f ∈ inj(A,B)
      using ord_iso_is_bij bij_is_inj by simp
    moreover
    from A1 A2 A4 have
      ⟨f(x), f(y)⟩ ∈ R and ⟨f(y), f(x)⟩ ∈ R
      using ord_iso_apply by auto
    with A3 have f(x) = f(y) by (rule Fol1_L4)
    moreover from A2 A4 have x∈A  y∈A by auto
    ultimately have x=y by (rule inj_apply_equality)
  } then have ∀x y. ⟨x,y⟩ ∈ r ∧ ⟨y,x⟩ ∈ r ⟶ x=y by auto
  then show antisym(r) using imp_conj antisym_def
    by simp
qed
```

Order isomoprhisms preserve transitivity.

```
lemma ord_iso_pres_trans: assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: trans(R)
  shows trans(r)
proof -
  { fix x y z
    assume A4: ⟨x, y⟩ ∈ r   ⟨y, z⟩ ∈ r
    note A1
    moreover
    from A1 A2 A4 have
      ⟨f(x), f(y)⟩ ∈ R ∧ ⟨f(y), f(z)⟩ ∈ R
      using ord_iso_apply by auto
    with A3 have ⟨f(x),f(z)⟩ ∈ R by (rule Fol1_L3)
    moreover from A2 A4 have x∈A  z∈A by auto
    ultimately have ⟨x, z⟩ ∈ r using ord_iso_apply_conv
      by simp
  } then have  ∀ x y z. ⟨x, y⟩ ∈ r ∧ ⟨y, z⟩ ∈ r ⟶ ⟨x, z⟩ ∈ r
    by blast
  then show trans(r) by (rule Fol1_L2)
```

```
qed
```

Order isomorphisms preserve totality.

```
lemma ord_iso_pres_tot: assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A and A3: R  {is total on} B
  shows r  {is total on} A
proof -
  { fix x y
    assume A4: x∈A  y∈A  ⟨x,y⟩ ∉ r
    with A1 have ⟨f(x),f(y)⟩ ∉ R using ord_iso_apply_conv
      by auto
    moreover
    from A1 have f:A→B using ord_iso_is_bij bij_is_fun
      by simp
    with A3 A4 have ⟨f(x),f(y)⟩ ∈  R ∨ ⟨f(y),f(x)⟩ ∈  R
      using apply_funtype IsTotal_def by simp
    ultimately have ⟨f(y),f(x)⟩ ∈  R by simp
    with A1 A4 have ⟨y,x⟩ ∈ r using ord_iso_apply_conv
      by simp
  } then have ∀x∈A. ∀y∈A. ⟨x,y⟩ ∈ r ∨  ⟨y,x⟩ ∈ r
    by blast
  then show r  {is total on} A using IsTotal_def
    by simp
qed
```

Order isomorphisms preserve linearity.

```
lemma ord_iso_pres_lin: assumes f ∈ ord_iso(A,r,B,R) and
  r ⊆ A×A and IsLinOrder(B,R)
  shows IsLinOrder(A,r)
  using prems ord_iso_pres_antsym ord_iso_pres_trans ord_iso_pres_tot
    IsLinOrder_def by simp
```

If a relation is a linear order, then the relation induced on another set is by a bijection is also a linear order.

```
lemma ind_rel_pres_lin:
  assumes A1: f ∈ bij(A,B) and A2: IsLinOrder(B,R)
  shows IsLinOrder(A,InducedRelation(f,R))
proof -
  let r = InducedRelation(f,R)
  from A1 have f ∈ ord_iso(A,r,B,R) and r ⊆ A×A
    using bij_is_ord_iso domain_of_bij InducedRelation_def
    by auto
  with A2 show IsLinOrder(A,r) using ord_iso_pres_lin
    by simp
qed
```

The image by an order isomorphism of a bounded above and nonempty set is bounded above.

```
lemma ord_iso_pres_bound_above:
  assumes A1: f ∈ ord_iso(A,r,B,R) and A2: r ⊆ A×A and
  A3: IsBoundedAbove(C,r)   C≠0
  shows IsBoundedAbove(f(C),R)   f(C) ≠ 0
proof -
  from A3 obtain u where I: ∀x∈C. ⟨x,u⟩ ∈ r
    using IsBoundedAbove_def by auto
  from A1 have II: f:A→B using ord_iso_is_bij bij_is_fun
    by simp
  from A2 A3 have III: C⊆A using Order_ZF_3_L1A by blast
  from A3 obtain x where x∈C by auto
  with A2 I have IV: u∈A by auto
  { fix y assume y ∈ f(C)
    with II III obtain x where x∈C and y = f(x)
      using func_imagedef by auto
    with A1 I III IV have ⟨y,f(u)⟩ ∈ R
      using ord_iso_apply by auto
  } then have ∀y ∈ f(C).  ⟨y,f(u)⟩ ∈ R by simp
  then show IsBoundedAbove(f(C),R) by (rule Order_ZF_3_L10)
  from A3 II III show f(C) ≠ 0 using func1_1_L15A
    by simp
qed
```

Order isomorphisms preserve the property of having a minimum.

```
lemma ord_iso_pres_has_min:
  assumes A1: f ∈ ord_iso(A,r,B,R) and  A2: r ⊆ A×A and
  A3: C⊆A and A4: HasAminimum(R,f(C))
  shows HasAminimum(r,C)
proof -
  from A4 obtain m where
    I: m ∈ f(C) and II: ∀y ∈ f(C). ⟨m,y⟩ ∈ R
    using HasAminimum_def by auto
  let k = converse(f)(m)
  from A1 have III: f:A→B using ord_iso_is_bij bij_is_fun
    by simp
  from A1 have f ∈ inj(A,B) using ord_iso_is_bij bij_is_inj
    by simp
  with A3 I have IV: k ∈ C and V: f(k) = m
    using inj_inv_back_in_set by auto
  moreover
  { fix x assume A5: x∈C
    with A3 II III IV V have
      k ∈ A   x∈A  ⟨f(k),f(x)⟩ ∈ R
      using func_imagedef by auto
    with A1 have ⟨k,x⟩ ∈ r using ord_iso_apply_conv
      by simp
  } then have ∀x∈C.  ⟨k,x⟩ ∈ r by simp
  ultimately show HasAminimum(r,C) using HasAminimum_def by auto
qed
```

Order isomorhisms preserve the images of relations. In other words taking the image of a point by a relation commutes with the function.

```
lemma ord_iso_pres_rel_image:
  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and
  A3: a∈A
  shows f(r{a}) = R{f(a)}
proof
  from A1 have f:A→B using ord_iso_is_bij bij_is_fun
    by simp
  moreover from A2 A3 have I: r{a} ⊆ A by auto
  ultimately have I: f(r{a}) = {f(x). x ∈ r{a} }
    using func_imagedef by simp
  { fix y assume A4: y ∈ f(r{a})
    with I obtain x where
      x ∈ r{a} and II: y = f(x)
      by auto
    with A1 A2 have ⟨f(a),f(x)⟩ ∈ R using ord_iso_apply
      by auto
    with II have y ∈  R{f(a)} by auto
  } then show  f(r{a}) ⊆ R{f(a)} by auto
  { fix y assume A5: y ∈ R{f(a)}
    let x = converse(f)(y)
    from A2 A5 have
      ⟨f(a),y⟩ ∈ R  f(a) ∈ B  and IV: y∈B
      by auto
    with A1 have III: ⟨converse(f)(f(a)),x⟩ ∈ r
      using ord_iso_converse by simp
    moreover from A1 A3 have converse(f)(f(a)) = a
      using ord_iso_is_bij left_inverse_bij by blast
    ultimately have f(x) ∈ {f(x). x ∈  r{a} }
      by auto
    moreover from A1 IV have f(x) = y
      using ord_iso_is_bij right_inverse_bij by blast
    moreover from A1 I have f(r{a}) = {f(x). x ∈  r{a} }
      using ord_iso_is_bij bij_is_fun func_imagedef by blast
    ultimately have y ∈ f(r{a}) by simp
  } then show R{f(a)} ⊆ f(r{a}) by auto
qed
```

Order isomorphisms preserve collections of upper bounds.

```
lemma ord_iso_pres_up_bounds:
  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and
  A3: C⊆A
  shows {f(r{a}). a∈C} = {R{b}. b ∈ f(C)}
proof
  from A1 have T: f:A→B
      using ord_iso_is_bij bij_is_fun by simp
```

```
{ fix Y assume Y ∈ {f(r{a}). a∈C}
    then obtain a where I: a∈C and II: Y = f(r{a})
      by auto
    from A3 I have a∈A by auto
    with A1 A2 have f(r{a}) = R{f(a)}
      using ord_iso_pres_rel_image by simp
    moreover from A3 T I have f(a) ∈ f(C)
      using func_imagedef by auto
    ultimately have f(r{a}) ∈ { R{b}. b ∈ f(C) }
      by auto
    with II have Y ∈ { R{b}. b ∈ f(C) } by simp
  } then show {f(r{a}). a∈C} ⊆ {R{b}. b ∈ f(C)}
    by blast
  { fix Y assume Y ∈ {R{b}. b ∈ f(C)}
    then obtain b where III: b ∈ f(C) and IV: Y = R{b}
      by auto
    with A3 T obtain a where V: a∈C and b = f(a)
      using func_imagedef by auto
    with A3 IV have a∈A and Y = R{f(a)} by auto
    with A1 A2 have Y = f(r{a})
      using ord_iso_pres_rel_image by simp
    with V have Y ∈ {f(r{a}). a∈C} by auto
  } then show {R{b}. b ∈ f(C)} ⊆ {f(r{a}). a∈C}
    by auto
qed
```

The image of the set of upper bounds is the set of upper bounds of the image.

```
lemma ord_iso_pres_min_up_bounds:
  assumes A1: f ∈ ord_iso(A,r,B,R) and  A2: r ⊆ A×A  R ⊆ B×B and
  A3: C⊆A and A4: C≠0
  shows f(⋂a∈C. r{a}) = (⋂b∈f(C). R{b})
proof -
  from A1 have f ∈ inj(A,B)
    using ord_iso_is_bij bij_is_inj by simp
  moreover note A4
  moreover from A2 A3 have ∀a∈C. r{a} ⊆ A by auto
  ultimately have
    f(⋂a∈C. r{a}) = ( ⋂a∈C. f(r{a}) )
    using inj_image_of_Inter by simp
  also from A1 A2 A3 have
    ( ⋂a∈C. f(r{a}) ) = ( ⋂b∈f(C). R{b} )
    using ord_iso_pres_up_bounds by simp
  finally show f(⋂a∈C. r{a}) = (⋂b∈f(C). R{b})
    by simp
qed
```

Order isomorphisms preserve completeness.

```
lemma ord_iso_pres_compl:
```

```
  assumes A1: f ∈ ord_iso(A,r,B,R) and
  A2: r ⊆ A×A  R ⊆ B×B and A3: R {is complete}
  shows r {is complete}
proof -
  { fix C
    assume A4: IsBoundedAbove(C,r)  C≠0
    with A1 A2 A3 have
      HasAminimum(R,⋂b ∈ f(C). R{b})
      using ord_iso_pres_bound_above IsComplete_def
      by simp
    moreover
    from A2 A4 have I: C ⊆ A using Order_ZF_3_L1A
      by blast
    with A1 A2 A4 have f(⋂a∈C. r{a}) = (⋂b∈f(C). R{b})
      using ord_iso_pres_min_up_bounds by simp
    ultimately have HasAminimum(R,f(⋂a∈C. r{a}))
      by simp
    moreover
    from A2 A4 have C≠0 and ∀a∈C. r{a} ⊆ A by auto
    then have ( ⋂a∈C. r{a} ) ⊆ A using ZF1_1_L7 by simp
    moreover note A1 A2
    ultimately have HasAminimum(r, ⋂a∈C. r{a} )
      using ord_iso_pres_has_min by simp
  } then show r {is complete} using IsComplete_def
    by simp
qed
```

If the original relation is complete, then the induced one is complete.

```
lemma ind_rel_pres_compl: assumes A1: f ∈ bij(A,B)
  and A2: R ⊆ B×B and A3: R {is complete}
  shows InducedRelation(f,R) {is complete}
proof -
  let r = InducedRelation(f,R)
  from A1 have f ∈ ord_iso(A,r,B,R)
    using bij_is_ord_iso by simp
  moreover from A1 A2 have r ⊆ A×A
    using bij_is_fun ind_rel_domain by simp
  moreover note A2 A3
  ultimately show r {is complete}
    using ord_iso_pres_compl by simp
qed



end
```

7 EquivClass1.thy

theory `EquivClass1` **imports** `EquivClass func_ZF ZF1`

begin

In this theory file we extend the work on equivalence relations done in the standard Isabelle's EquivClass.thy file. The problem that we have with the EquivClass.thy is that the notions congruent and congruent2 are defined for meta-functions rather then ZF - functions (subsets of Cartesian products). This causes inflexibility (that is typical for typed set theories) in making the notions depend on additional parameters For example the congruent2 there takes $[i, [i, i] => i]$ as parameters, that is the second parameter is a meta-function that takes two sets and results in a set. So, when our function depends on additional parameters, (for example the function we want to be congruent depends on a group and we want to show that for all groups the function is congruent) there is no easy way to use that notion. The ZF functions are sets and there is no problem if in actual application this set depends on some parameters.

7.1 Congruent functions and projections on the quotient

First we define the notion of function that maps equivalent elements to equivalent values. We use similar names as in the original `EquivClass.thy` file to indicate the conceptual correspondence of the notions. Then we define the projection of a function onto the quotient space. We will show that if the function is congruent the projection is a mapping from the quotient space into itself. In standard math the condion that the function is congruent allows to show that the value of the projection does not depend on the choice of elements that represent the equivalence classes. We set up things a little differently to avoid making choices.

```
constdefs
  Congruent(r,f) ≡
  (∀x y. <x,y> ∈ r  ⟶ <f(x),f(y)> ∈ r)

  ProjFun(A,r,f) ≡
  {<c,d> ∈ (A//r)×(A//r). (⋃x∈c. r{f(x)}) = d}
```

Elements of equivalence classes belong to the set.

```
lemma EquivClass_1_L1:
  assumes A1: equiv(A,r) and A2: C ∈ A//r and A3: x∈C
  shows x∈A
proof -
  from A2 have C ⊆ ⋃ (A//r) by auto
  with A1 A3 show x∈A
    using Union_quotient by auto
```

qed

The image of a subset of X under projection is a subset of A/r.

```
lemma EquivClass_1_L1A:
  assumes A⊆X shows {r{x}. x∈A} ⊆ X//r
  using prems quotientI by auto
```

If an element belongs to an equivalence class, then its image under relation is this equivalence class.

```
lemma EquivClass_1_L2:
  assumes A1: equiv(A,r)  C ∈ A//r and A2: x∈C
  shows r{x} = C
proof -
  from A1 A2 have x ∈ r{x}
    using EquivClass_1_L1  equiv_class_self by simp
  with A2 have T1:r{x}∩C ≠ 0 by auto
  from A1 A2 have r{x} ∈ A//r
    using EquivClass_1_L1 quotientI by simp
  with A1 T1 show thesis
    using quotient_disj by blast
qed
```

Elements that belong to the same equivalence class are equivalent.

```
lemma EquivClass_1_L2A:
  assumes equiv(A,r)  C ∈ A//r  x∈C  y∈C
  shows <x,y> ∈ r
  using prems EquivClass_1_L2 EquivClass_1_L1 equiv_class_eq_iff
  by simp
```

Every x is in the class of y, then they are equivalent.

```
lemma EquivClass_1_L2B:
  assumes A1: equiv(A,r) and A2: y∈A and A3: x ∈ r{y}
  shows <x,y> ∈ r
proof -
  from A2 have  r{y} ∈ A//r
    using quotientI by simp
  with A1 A3 show thesis using
    EquivClass_1_L1 equiv_class_self equiv_class_nondisjoint by blast
qed
```

If a function is congruent then the equivalence classes of the values that come from the arguments from the same class are the same.

```
lemma EquivClass_1_L3:
  assumes A1: equiv(A,r) and A2: Congruent(r,f)
  and A3: C ∈ A//r  x∈C  y∈C
  shows r{f(x)} = r{f(y)}
proof -
  from A1 A3 have <x,y> ∈ r
```

```
    using EquivClass_1_L2A by simp
  with A2 have  <f(x),f(y)> ∈ r
    using Congruent_def by simp
  with A1 show thesis using equiv_class_eq by simp
qed
```

The values of congruent functions are in the space.

```
lemma EquivClass_1_L4:
  assumes A1: equiv(A,r) and A2: C ∈ A//r  x∈C
  and A3: Congruent(r,f)
  shows f(x) ∈ A
proof -
  from A1 A2 have x∈A
    using EquivClass_1_L1 by simp
  with A1 have <x,x> ∈ r
    using equiv_def refl_def by simp
  with A3 have  <f(x),f(x)> ∈ r
    using Congruent_def by simp
  with A1 show thesis using equiv_type by auto
qed
```

Equivalence classes are not empty.

```
lemma EquivClass_1_L5:
  assumes A1: refl(A,r) and A2: C ∈ A//r
  shows C≠0
proof -
  from A2 obtain x where D1: C = r{x} and D2: x∈A
    using quotient_def by auto
  from D2 A1 have x ∈ r{x} using refl_def by auto
  with D1 show thesis by auto
qed
```

To avoid using an axiom of choice, we define the projection using the expression $\bigcup_{x\in C} r(\{f(x)\})$. The next lemma shows that for congruent function this is in the quotient space A/r.

```
lemma EquivClass_1_L6:
  assumes A1: equiv(A,r) and A2: Congruent(r,f)
  and A3:C ∈ A//r
  shows (⋃x∈C. r{f(x)}) ∈ A//r
proof -
  from A1 A3 have C≠0
    using equiv_def EquivClass_1_L5 by auto
  moreover from A2 A3 A1 have ∀x∈C. r{f(x)} ∈ A//r
    using EquivClass_1_L4 quotientI by auto
  moreover from A1 A2 A3 have
    ∀x y. x∈C ∧ y∈C ⟶ r{f(x)} = r{f(y)}
    using EquivClass_1_L3 by blast
  ultimately show thesis by (rule ZF1_1_L2)
```

qed

Congruent functions can be projected.

lemma EquivClass_1_T1:
  assumes equiv(A,r) Congruent(r,f)
  shows ProjFun(A,r,f) ∈ A//r → A//r
  using prems EquivClass_1_L6 ProjFun_def func1_1_L11A
  by simp

We now define congruent functions of two variables. Congruent2 corresponds to congruent2 in EquivClass.thy, but uses ZF-functions rather than meta-functions.

constdefs
  Congruent2(r,f) ≡
  (∀x1 x2 y1 y2. <x1,x2> ∈ r ∧ <y1,y2> ∈ r ⟶
  <f<x1,y1>,f<x2,y2> > ∈ r)

  ProjFun2(A,r,f) ≡
  {<p,d> ∈ ((A//r)×(A//r))×(A//r) .
  (⋃ z ∈ fst(p)×snd(p). r{f(z)}) = d}

The following lemma is a two-variables equivalent of EquivClass_1_L3.

lemma EquivClass_1_L7:
  assumes A1: equiv(A,r) **and** A2: Congruent2(r,f)
  and A3: C1 ∈ A//r C2 ∈ A//r
  and A4: z1 ∈ C1×C2 z2 ∈ C1×C2
  shows r{f(z1)} = r{f(z2)}
proof -
  from A4 **obtain** x1 y1 x2 y2 **where**
    x1∈C1 **and** y1∈C2 **and** D1:z1 = <x1,y1> **and**
    x2∈C1 **and** y2∈C2 **and** D2:z2 = <x2,y2>
    by auto
  with A1 A3 **have** <x1,x2> ∈ r **and** <y1,y2> ∈ r
    using EquivClass_1_L2A **by** auto
  with A2 **have** <f<x1,y1>,f<x2,y2> > ∈ r
    using Congruent2_def **by** simp
  with A1 D1 D2 **show** thesis **using** equiv_class_eq **by** simp
qed

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

lemma EquivClass_1_L8:
  assumes A1: equiv(A,r) **and** A2: C1 ∈ A//r **and** A3: C2 ∈ A//r
  and A4: z ∈ C1×C2 **and** A5: Congruent2(r,f)
  shows f(z) ∈ A
proof -
  from A4 **obtain** x y **where** x∈C1 **and** y∈C2 **and** D1:z = <x,y>
    by auto
  with A1 A2 A3 **have** x∈A **and** y∈A

```
    using EquivClass_1_L1 by auto
  with A1 A4 have <x,x> ∈ r and <y,y> ∈ r
    using equiv_def refl_def by auto
  with A5 have <f<x,y>, f<x,y> > ∈ r
    using Congruent2_def by simp
  with A1 D1 show thesis using equiv_type by auto
qed
```

The values of congruent functions are in the space. Note that although this lemma is intended to be used with functions, we don't need to assume that we f is a function.

```
lemma EquivClass_1_L8A:
  assumes A1: equiv(A,r) and A2: x∈A  y∈A
  and A3: Congruent2(r,f)
  shows f<x,y> ∈ A
proof -
  from A1 A2 have r{x} ∈ A//r r{y} ∈ A//r
    <x,y> ∈ r{x}×r{y}
    using equiv_class_self quotientI by auto
  with A1 A3 show thesis using EquivClass_1_L8 by simp
qed
```

The following lemma is a two-variables equivalent of EquivClass_1_L6.

```
lemma EquivClass_1_L9:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: p ∈ (A//r)×(A//r)
  shows (⋃ z ∈ fst(p)×snd(p). r{f(z)}) ∈ A//r
proof -
  from A3 have D1:fst(p) ∈ A//r and D2:snd(p) ∈ A//r
    by auto
  with A1 A2 have
    T1:∀z ∈ fst(p)×snd(p). f(z) ∈ A
    using EquivClass_1_L8 by simp
  from A3 A1 have fst(p)×snd(p) ≠ 0
    using equiv_def EquivClass_1_L5 Sigma_empty_iff
    by auto
  moreover from A1 T1 have
    ∀z ∈ fst(p)×snd(p). r{f(z)} ∈ A//r
    using quotientI by simp
  moreover from A1 A2 D1 D2 have
    ∀z1 z2. z1 ∈ fst(p)×snd(p) ∧ z2 ∈ fst(p)×snd(p) ⟶
    r{f(z1)} = r{f(z2)}
    using EquivClass_1_L7 by blast
   ultimately show thesis by (rule ZF1_1_L2)
qed
```

Congruent functions of two variables can be projected.

```
theorem EquivClass_1_T1:
```

```
  assumes equiv(A,r)  Congruent2(r,f)
  shows ProjFun2(A,r,f) ∈ (A//r)×(A//r) → A//r
  using prems EquivClass_1_L9 ProjFun2_def func1_1_L11A by simp
```

We define the projection on the quotient space as a function that takes an element of A and assigns its equivalence class in A/r.

```
constdefs
  Proj(A,r) ≡ {<x,c> ∈ A×(A//r). r{x} = c}
```

The projection diagram commutes. I wish I knew how to draw this diagram in LaTeX.

```
lemma EquivClass_1_L10: assumes A1: equiv(A,r) and A2: Congruent2(r,f)

  and A3: x∈A  y∈A
  shows ProjFun2(A,r,f)<r{x},r{y}> = r{f<x,y>}
proof -
  from A3 A1 have r{x} × r{y} ≠ 0
    using quotientI equiv_def EquivClass_1_L5 Sigma_empty_iff
    by auto
  moreover have
    ∀z ∈ r{x}×r{y}.  r{f(z)} = r{f<x,y>}
  proof
    fix z assume A4:z ∈ r{x}×r{y}
    from A1 A3 have
      r{x} ∈ A//r r{y} ∈ A//r
      <x,y> ∈ r{x}×r{y}
      using quotientI equiv_class_self by auto
    with A1 A2 A4 show
      r{f(z)} = r{f<x,y>}
      using EquivClass_1_L7 by blast
  qed
  ultimately have
    (⋃z ∈ r{x}×r{y}. r{f(z)}) =  r{f<x,y>}
    by (rule ZF1_1_L1)
  moreover from A3 A1 A2 have
    ProjFun2(A,r,f)<r{x},r{y}> =
    (⋃z ∈ r{x}×r{y}. r{f(z)})
    using quotientI EquivClass_1_T1 ProjFun2_def func1_1_L11B
    by simp
  ultimately show thesis by simp
qed
```

7.2 Projecting commutative, associative and distributive operations.

In this section we show that if the operations are congruent with respect to an equivalence relation then the projection to the quotient space preserves commutativity, associativity and distributivity.

The projection of commutative operation is commutative.

```
lemma EquivClass_2_L1: assumes
  A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f {is commutative on} A
  and A4: c1 ∈ A//r  c2 ∈ A//r
  shows ProjFun2(A,r,f) <c1,c2> =  ProjFun2(A,r,f)<c2,c1>
proof -
  from A4 obtain x y where D1:
    c1 = r{x} c2 = r{y}
    x∈A y∈A
    using quotient_def by auto
  with A1 A2 have ProjFun2(A,r,f) <c1,c2> =  r{f<x,y>}
    using EquivClass_1_L10 by simp
  also from A3 D1 have
    r{f<x,y>} = r{f<y,x>}
    using IsCommutative_def by simp
  also from A1 A2 D1 have
    r{f<y,x>} = ProjFun2(A,r,f) <c2,c1>
    using EquivClass_1_L10 by simp
  finally show thesis by simp
qed
```

The projection of commutative operation is commutative.

```
theorem EquivClass_2_T1:
  assumes equiv(A,r) and Congruent2(r,f)
  and f {is commutative on} A
  shows ProjFun2(A,r,f) {is commutative on} A//r
  using prems IsCommutative_def EquivClass_2_L1 by simp
```

The projection of an associative operation is associative.

```
lemma EquivClass_2_L2:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f {is associative on} A
  and A4: c1 ∈ A//r  c2 ∈ A//r  c3 ∈ A//r
  and A5: g = ProjFun2(A,r,f)
  shows g⟨g<c1,c2>,c3⟩ = g⟨c1,g<c2,c3>⟩
proof -
  from A4 obtain x y z where D1:
    c1 = r{x} c2 = r{y} c3 = r{z}
    x∈A y∈A z∈A
    using quotient_def by auto
  with A3 have T1:f<x,y> ∈ A f<y,z> ∈ A
    using IsAssociative_def apply_type by auto
  with A1 A2 D1 A5 have
    g⟨g<c1,c2>,c3⟩ =  r{f<f<x,y>,z>}
    using EquivClass_1_L10 by simp
  also from D1 A3 have
    ... = r{f<x,f<y,z> >}
    using IsAssociative_def by simp
```

```
  also from T1 A1 A2 D1 A5 have
    ... = g⟨c1,g<c2,c3>⟩
    using EquivClass_1_L10 by simp
  finally show thesis by simp
qed
```

The projection of an associative operation is associative on the quotient.

```
theorem EquivClass_2_T2:
  assumes A1: equiv(A,r) and A2: Congruent2(r,f)
  and A3: f {is associative on} A
  shows ProjFun2(A,r,f) {is associative on} A//r
proof -
  let g = ProjFun2(A,r,f)
  from A1 A2 have
    g ∈ (A//r)×(A//r) → A//r
    using EquivClass_1_T1 by simp
  moreover from A1 A2 A3 have
    ∀c1 ∈ A//r.∀c2 ∈ A//r.∀c3 ∈ A//r.
    g<g<c1,c2>,c3> = g< c1,g<c2,c3> >
    using EquivClass_2_L2 by simp
  ultimately show thesis
    using IsAssociative_def by simp
qed
```

The essential condition to show that distributivity is preserved by projections to quotient spaces, provided both operations are congruent with respect to the equivalence relation.

```
lemma EquivClass_2_L3:
  assumes A1: IsDistributive(X,A,M)
  and A2: equiv(X,r)
  and A3: Congruent2(r,A) Congruent2(r,M)
  and A4: a ∈ X//r b ∈ X//r c ∈ X//r
  and A5: Ap = ProjFun2(X,r,A) Mp = ProjFun2(X,r,M)
  shows Mp⟨a,Ap<b,c>⟩ = Ap⟨ Mp<a,b>,Mp<a,c>⟩ ∧
  Mp⟨ Ap<b,c>,a ⟩ = Ap⟨ Mp<b,a>,Mp<c,a>⟩
proof
  from A4 obtain x y z where x∈X y∈X z∈X
    a = r{x} b = r{y} c = r{z}
    using quotient_def by auto
  with A1 A2 A3 A5 show
    Mp⟨a,Ap<b,c>⟩ = Ap⟨ Mp<a,b>,Mp<a,c>⟩
    Mp⟨ Ap<b,c>,a ⟩ = Ap⟨ Mp<b,a>,Mp<c,a>⟩
    using EquivClass_1_L8A EquivClass_1_L10 IsDistributive_def
    by auto
qed
```

Distributivity is preserved by projections to quotient spaces, provided both operations are congruent with respect to the equivalence relation.

```
lemma EquivClass_2_L4: assumes A1: IsDistributive(X,A,M)
```

```
  and A2: equiv(X,r)
  and A3: Congruent2(r,A) Congruent2(r,M)
  shows IsDistributive(X//r,ProjFun2(X,r,A),ProjFun2(X,r,M))
proof-
 let Ap = ProjFun2(X,r,A)
 let Mp = ProjFun2(X,r,M)
 from A1 A2 A3 have
   ∀a∈X//r.∀b∈X//r.∀c∈X//r.
   Mp< a,Ap<b,c> > = Ap< Mp<a,b>,Mp<a,c> > ∧
   Mp< Ap<b,c>,a > = Ap< Mp<b,a>,Mp<c,a> >
   using EquivClass_2_L3 by simp
 then show thesis using IsDistributive_def by simp
qed
```

7.3 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 equivalemce classes. This makes them useful as a tool to define subsets of the quoutient 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 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.

```
constdefs
  IsSaturated(r,A) ≡ A = r-(r(A))
```

For equivalence relations a set is saturated iff it is an image of itself.

```
lemma EquivClass_3_L1: assumes A1: equiv(X,r)
  shows IsSaturated(r,A) ⟷ A = r(A)
proof
  assume A2: IsSaturated(r,A)
  then have A = (converse(r) O r)(A)
    using IsSaturated_def vimage_def image_comp
    by simp
  also from A1 have ... = r(A)
    using equiv_comp_eq by simp
  finally show A = r(A) by simp
next assume A = r(A)
  with A1 have A = (converse(r) O r)(A)
    using equiv_comp_eq by simp
  also have ... =  r-(r(A))
    using vimage_def image_comp by simp
  finally have A =  r-(r(A)) by simp
```

```
  then show IsSaturated(r,A) using IsSaturated_def
    by simp
qed
```

For equivalence relations sets are contained in their images.

```
lemma EquivClass_3_L2: assumes A1: equiv(X,r) and A2: A⊆X
  shows A ⊆ r(A)
proof
  fix a assume A3: a∈A
  with A1 A2 have a ∈ r{a}
    using equiv_class_self by auto
  with A3 show a ∈ r(A) by auto
qed
```

The next lemma shows that if "$\sim$" is an equivalence relation and a set A is such that $a \in A$ and $a \sim b$ implies $b \in A$, then A is saturated with respect to the relation.

```
lemma EquivClass_3_L3: assumes A1: equiv(X,r)
  and A2: r ⊆ X×X and A3: A⊆X
  and A4: ∀x∈A. ∀y∈X. ⟨x,y⟩ ∈ r ⟶ y∈A
  shows IsSaturated(r,A)
proof -
  from A2 A4 have r(A) ⊆ A
    using image_iff by blast
  moreover from A1 A3 have A ⊆ r(A)
    using EquivClass_3_L2 by simp
  ultimately have A = r(A) by auto
  with A1 show IsSaturated(r,A) using EquivClass_3_L1
    by simp
qed
```

If $A \subseteq X$ and A is saturated and $x \sim y$, then $x \in A$ iff $y \in A$. Here we we show only one direction.

```
lemma EquivClass_3_L4: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: ⟨x,y⟩ ∈ r
  and A5: x∈X  y∈A
  shows x∈A
proof -
  from A1 A5 have x ∈ r{x}
    using equiv_class_self by simp
  with A1 A3 A4 A5 have x ∈ r(A)
    using equiv_class_eq equiv_class_self
    by auto
  with A1 A2 show x∈A
    using EquivClass_3_L1 by simp
qed
```

If $A \subseteq X$ and A is saturated and $x \sim y$, then $x \in A$ iff $y \in A$.

```
lemma EquivClass_3_L5: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: x∈X  y∈X
  and A5: ⟨x,y⟩ ∈ r
  shows x∈A ⟷ y∈A
proof
  assume y∈A
  with prems show x∈A using EquivClass_3_L4
    by simp
next assume A6: x∈A
  from A1 A5 have ⟨y,x⟩ ∈ r
    using equiv_is_sym by blast
  with A1 A2 A3 A4 A6 show y∈A
    using EquivClass_3_L4 by simp
qed
```

If A is saturated then $x \in A$ iff its class is in the projection of A.

```
lemma EquivClass_3_L6: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X and A4: x∈X
  and A5: B = {r{x}. x∈A}
  shows x∈A ⟷ r{x} ∈ B
proof
  assume x∈A
  with A5 show r{x} ∈ B by auto
next assume r{x} ∈ B
  with A5 obtain y where I: y ∈ A and r{x} = r{y}
    by auto
  with A1 A3 have ⟨x,y⟩ ∈ r
    using eq_equiv_class by auto
  with A1 A2 A3 A4 I show x∈A
    using EquivClass_3_L4 by simp
qed
```

A technical lemma involving a projection of a saturated set and a logical epression with exclusive or.

```
lemma EquivClass_3_L7: assumes A1: equiv(X,r)
  and A2: IsSaturated(r,A) and A3: A⊆X
  and A4: x∈X  y∈X
  and A5: B = {r{x}. x∈A}
  and A6: (x∈A) Xor (y∈A)
  shows (r{x} ∈ B)  Xor (r{y} ∈ B)
  using prems EquivClass_3_L6 by simp
```

```
end
```

8 Finite1.thy

```
theory Finite1 imports Finite func1 ZF1

begin
```

8.1 Finite powerset

Intersection of a collection is contained in every element of the collection.

```
lemma ZF11: assumes A: A ∈ M shows ⋂M ⊆ A
proof
  fix x
  assume A1: x ∈ ⋂M
  from A1 A show x ∈ A ..
qed
```

Intersection of a nonempty collection M of subsets of X is a subset of X.

```
lemma ZF12: assumes A1: ∀A∈ M. A⊆X and A2:M≠0
  shows (⋂ M) ⊆ X
proof -
 from A2 have ∀ A∈ M. (⋂ M ⊆ A) using ZF11 by simp
 with A1 A2 show  (⋂ M) ⊆ X by fast
qed
```

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.

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

In Topology_ZFTopology_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.

```
constdefs
  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 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 (cases M=0)
  assume A7: M=0 show Prfin(T, A, cons(W, M))
  proof-
    from A1 A obtain U where A5: U∈T and A6:W=U∩A by fast
    let N = {U}
    from A5 have T1: N ∈ Fin(T) by simp
    from A7 A6 have T2:∀V∈ cons(W,M). ∃ U∈N. V=U∩A by simp
    from A7 T1 T2 show Prfin(T, A, cons(W, M))
      using Prfin_def by auto
  qed
next
  assume A8:M≠0 show Prfin(T, A, cons(W, M))
  proof-
    from A1 A obtain U where A5: U∈T and A6:W=U∩A by fast
    from A8 A4 obtain N0
      where A9: N0∈ Fin(T)
      and A10: ∀V∈ M. ∃ U0∈ N0. (V = U0∩A)
      using Prfin_def by auto
    let N = cons(U,N0)
    from A5 A9 have N ∈ Fin(T) by simp
    moreover from A10 A6 have ∀V∈ cons(W,M). ∃ U∈N. V=U∩A by simp
    ultimately have ∃ N∈ Fin(T).∀V∈ cons(W,M). ∃ U∈N. V=U∩A by auto
    with A8 show Prfin(T, A, cons(W, M))
      using Prfin_def by simp
  qed
qed
```

Now we are ready to prove the statement we need.

```
theorem FinRestr0: assumes A: ∀ V∈ TA. ∃ U∈ T. V=U∩A
  shows ∀ M∈ Fin(TA). Prfin(T,A,M)
proof
  fix M
  assume A1: M∈ Fin(TA)
  have Prfin(T,A,0) using Prfin_def by simp
  with A1 show Prfin(T,A,M) using ind_step by (rule Fin_induct)
qed
```

This is a different form of the above theorem:

```
theorem ZF1FinRestr:
  assumes A1:M∈ Fin(TA) and A2: M≠0
  and A3: ∀ V∈ TA. ∃ U∈ T. V=U∩A
  shows ∃N∈ Fin(T). (∀V∈ M. ∃ U∈ N. (V = U∩A)) ∧ N≠0
```

```
proof -
  from A3 A1 have Prfin(T,A,M) using FinRestr0 by blast
  then have ∃N∈ Fin(T). ∀V∈ M. ∃ U∈ N. (V = U∩A)
    using A2 Prfin_def by simp
  then obtain N where
    D1:N∈ Fin(T) ∧ (∀V∈ M. ∃ U∈ N. (V = U∩A)) by auto
  with A2 have N≠0 by auto
  with D1 show thesis by auto
qed
```

Purely technical lemma used in Topology_ZF_1 to show that if a topology is T_2, then it is T_1.

```
lemma Finite1_L2:
  assumes A:∃U V. (U∈T ∧ V∈T ∧ x∈U ∧ y∈V ∧ U∩V=0)
  shows ∃U∈T. (x∈U ∧ y∉U)
proof -
  from A obtain U V where D1:U∈T ∧ V∈T ∧ x∈U ∧ y∈V ∧ U∩V=0 by auto
  with D1 show thesis by auto
qed
```

A collection closed with respect to taking a union of two sets is closed under taking finite unions. Proof by induction with the induction step formulated in a separate lemma.

The induction step:

```
lemma Finite1_L3_IndStep:
  assumes A1:∀A B. ((A∈C ∧ B∈C) ⟶ A∪B∈C)
  and A2: A∈C and A3:N∈Fin(C) and A4:A∉N and A5:⋃N ∈ C
  shows ⋃cons(A,N) ∈ C
proof -
  have ⋃ cons(A,N) = A∪ ⋃N by blast
  with A1 A2 A5 show thesis by simp
qed
```

The lemma:

```
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 -
  from A1 have ⋃0 ∈ C by simp
  with A3 show ⋃N∈ C using Finite1_L3_IndStep by (rule Fin_induct)
qed
```

A collection closed with respect to taking a intersection of two sets is closed under taking finite intersections. Proof by induction with the induction step formulated in a separate lemma. This is sligltly more involved than the union case in Finite1_L3, because the intersection of empty collection is undefined (or should be treated as such). To simplify notation we define the property to be proven for finite sets as a constdef.

```
constdefs
  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 (cases N=0)
  assume A6:N=0 show IntPr(T,cons(A,N))
  proof-
    from A6 A2 show IntPr(T, cons(A, N)) using IntPr_def by simp
  qed
  next
  assume A7:N≠0 show IntPr(T, cons(A, N))
  proof -
    from A7 A5 A2 A1 have ⋂N ∩ A ∈ T using IntPr_def by simp
    moreover from A7 have ⋂cons(A, N) = ⋂N ∩ A by auto
    ultimately show IntPr(T, cons(A, N)) using IntPr_def by simp
  qed
qed
```

The lemma.

```
lemma Finite1_L4:
  assumes A1:∀A B. A∈T ∧ B∈T ⟶ A∩B ∈ T
  and A2:N∈Fin(T)
  shows IntPr(T,N)
proof -
  have IntPr(T,0) using IntPr_def by simp
  with A2 show IntPr(T,N)  using Finite1_L4_IndStep
    by (rule Fin_induct)
qed
```

Next is a restatement of the above lemma that does not depend on the IntPr meta-function.

```
lemma Finite1_L5:
  assumes A1: ∀A B. ((A∈T ∧ B∈T) ⟶ A∩B∈T)
  and A2:N≠0 and A3:N∈Fin(T)
  shows ⋂N ∈ T
proof -
  from A1 A3 have IntPr(T,N) using Finite1_L4 by simp
  with A2 show thesis using IntPr_def by simp
qed
```

The images of finite subsets by a meta-function are finite. For example in topology if we have a finite collection of sets, then closing each of them results in a finite collection of closed sets. This is a very useful lemma with many unexpected applications. The proof is by induction.

The induction step:

```
lemma Finite1_L6_IndStep:
  assumes ∀V∈B. K(V)∈C
  and U∈B and N∈Fin(B) and U∉N and {K(V). V∈N}∈Fin(C)
  shows {K(V). V∈cons(U,N)} ∈ Fin(C)
  using prems by simp
```

The lemma:

```
lemma Finite1_L6: assumes A1:∀V∈B. K(V)∈C and A2:N∈Fin(B)
  shows {K(V). V∈N} ∈ Fin(C)
proof -
  have {K(V). V∈0}∈Fin(C) by simp
  with A2 show thesis using Finite1_L6_IndStep by (rule Fin_induct)
qed
```

The image of a finite set is finite.

```
lemma Finite1_L6A: assumes A1: f:X→Y and A2: N ∈ Fin(X)
  shows f(N) ∈ Fin(Y)
proof -
  from A1 have ∀x∈X. f(x) ∈ Y
    using apply_type by simp
  moreover from A2 have N∈Fin(X) .
  ultimately have {f(x). x∈N} ∈ Fin(Y)
    by (rule Finite1_L6)
  with A1 A2 show thesis
    using FinD  func_imagedef by simp
qed
```

If the set defined by a meta-function is finite, then every set defined by a composition of this meta function with another one is finite.

```
lemma Finite1_L6B:
  assumes A1: ∀x∈X. a(x) ∈ Y and A2: {b(y).y∈Y} ∈ Fin(Z)
  shows {b(a(x)).x∈X} ∈ Fin(Z)
proof -
  from A1 have {b(a(x)).x∈X} ⊆ {b(y).y∈Y} by auto
  with A2 show thesis using Fin_subset_lemma by blast
qed
```

If the set defined by a meta-function is finite, then every set defined by a composition of this meta function with another one is finite.

```
lemma Finite1_L6C:
  assumes A1: ∀y∈Y. b(y) ∈ Z and A2: {a(x). x∈X} ∈ Fin(Y)
  shows {b(a(x)).x∈X} ∈ Fin(Z)
proof -
  let N = {a(x). x∈X}
```

```
  from A1 A2 have {b(y). y ∈ N} ∈ Fin(Z)
    by (rule Finite1_L6)
  moreover have {b(a(x)). x∈X} = {b(y). y∈ N}
    by auto
  ultimately show thesis by simp
qed
```

Next we show an identity that is used to prove sufficiency of some condition for a collection of sets to be a base for a topology. Should be in ZF1.thy.

```
lemma Finite1_L8: assumes A1:∀U∈C. ∃A∈B. U = ⋃A
  shows ⋃⋃ {⋃{A∈B. U = ⋃A}. U∈C} = ⋃C
proof
  show ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A}) ⊆ ⋃C by blast
  show ⋃C ⊆ ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A})
  proof
    fix x assume A2:x ∈ ⋃C
    show x∈ ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A})
    proof -
      from A2 obtain U where D1:U∈C ∧ x∈U by auto
      with A1 obtain A where D2:A∈B ∧ U = ⋃A by auto
      from D1 D2 show x∈ ⋃(⋃U∈C. ⋃{A ∈ B . U = ⋃A}) by auto
    qed
  qed
qed
```

If an intersection of a collection is not empty, then the collection is not empty. We are (ab)using the fact the the intesection of empty collection is defined to be empty and prove by contradiction. Should be in ZF1.thy

```
lemma Finite1_L9: assumes A1:⋂A ≠ 0 shows A≠0
proof (rule ccontr)
  assume A2: ¬ A ≠ 0
  with A1 show False by simp
qed
```

Cartesian product of finite sets is finite.

```
lemma Finite1_L12: assumes A1:A ∈ Fin(A) and A2:B ∈ Fin(B)
  shows A×B ∈ Fin(A×B)
proof -
  have T1:∀a∈A. ∀b∈B. {<a,b>} ∈ Fin(A×B) by simp
  have ∀a∈A. {{<a,b>}. b ∈ B} ∈ Fin(Fin(A×B))
  proof
    fix a assume A3:a ∈ A
    with T1 have  ∀b∈B. {<a,b>} ∈ Fin(A×B)
      by simp
    moreover from A2 have B ∈ Fin(B) .
    ultimately show {{<a,b>}. b ∈ B} ∈ Fin(Fin(A×B))
      by (rule Finite1_L6)
  qed
```

```
  then have ∀a∈A. ⋃ {{<a,b>}. b ∈ B} ∈ Fin(A×B)
    using Fin_UnionI by simp
  moreover have
    ∀a∈A. ⋃ {{<a,b>}. b ∈ B} = {a}× B by blast
  ultimately have ∀a∈A. {a}× B ∈ Fin(A×B) by simp
  moreover from A1 have A ∈ Fin(A) .
  ultimately have {{a}× B. a∈A} ∈ Fin(Fin(A×B))
    by (rule Finite1_L6)
  then have ⋃{{a}× B. a∈A} ∈ Fin(A×B)
    using Fin_UnionI by simp
  moreover have ⋃{{a}× B. a∈A} = A×B by blast
  ultimately show thesis by simp
qed
```

We define the characterisic meta-function that is the identity on a set and assigns a default value everywhere else.

```
constdefs
  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 (cases A=0)
  assume A=0 then show A ∈ Fin(A) by simp
  next
  assume A2: A≠0 then obtain c where D1:c∈A
    by auto
  then have ∀x∈X. Characteristic(A,c,x) ∈ A
    using Characteristic_def by simp
  moreover from A1 have A ∈ Fin(X) .
  ultimately have
    {Characteristic(A,c,x). x∈A} ∈ Fin(A)
    by (rule Finite1_L6)
  moreover from D1 have
    {Characteristic(A,c,x). x∈A} = A
    using Characteristic_def by simp
  ultimately show A ∈ Fin(A) by simp
qed
```

Cartesian product of finite subsets is a finite subset of cartesian product.

```
lemma Finite1_L14: assumes A1:A ∈ Fin(X) B ∈ Fin(Y)
  shows A×B ∈ Fin(X×Y)
proof -
  from A1 have A×B ⊆ X×Y using FinD by auto
  then have Fin(A×B) ⊆ Fin(X×Y) using Fin_mono by simp
  moreover from A1 have A×B ∈ Fin(A×B)
    using Finite1_L13 Finite1_L12 by simp
  ultimately show thesis by auto
qed
```

The next lemma is needed in the Group_ZF_3 theory in a couple of places.

```
lemma Finite1_L15:
  assumes A1: {b(x). x∈A} ∈ Fin(B)  {c(x). x∈A} ∈ Fin(C)
  and A2: f : B×C→E
  shows {f<b(x),c(x)>. x∈A} ∈ Fin(E)
proof -
  from A1 have {b(x). x∈A}×{c(x). x∈A} ∈ Fin(B×C)
    using Finite1_L14 by simp
  moreover have
    {<b(x),c(x)>. x∈A} ⊆ {b(x). x∈A}×{c(x). x∈A}
    by blast
  ultimately have T0: {<b(x),c(x)>. x∈A} ∈ Fin(B×C)
    by (rule Fin_subset_lemma)
  with A2 have T1: f{<b(x),c(x)>. x∈A} ∈ Fin(E)
    using Finite1_L6A by auto
  from T0 have ∀x∈A. <b(x),c(x)> ∈ B×C
    using FinD by auto
  with A2 have
    f{<b(x),c(x)>. x∈A} = {f<b(x),c(x)>. x∈A}
    using func1_1_L17 by simp
  with T1 show thesis by simp
qed
```

Singletons are in the finite powerset.

```
lemma Finite1_L16: assumes x∈X shows {x} ∈ Fin(X)
  using prems emptyI consI by simp
```

A special case of Finite1_L15 where the second set is a singleton. Group_ZF_3 theory this corresponds to the situation where we multiply by a constant.

```
lemma Finite1_L16AA: assumes A1: {b(x). x∈A} ∈ Fin(B)
  and A2: c∈C and A3: f : B×C→E
  shows {f<b(x),c>. x∈A} ∈ Fin(E)
proof -
  from prems have
    ∀y∈B. f⟨y,c⟩ ∈ E
    {b(x). x∈A} ∈ Fin(B)
    using apply_funtype by auto
  then show thesis by (rule Finite1_L6C)
qed
```

In the IsarMathLib coding convention it is rather difficult to use results that take $\Longrightarrow$ (that is, another lemma) as one of the assumptions. It is easier to use a condition written with the first order implication ($\longrightarrow$). The next lemma is the induction step of the lemma about the first order induction.

```
lemma Finite1_L16A:
  assumes ∀A∈Fin(X).∀x∈X. x∉A ∧ P(A)⟶P(A∪{x})
  and x∈X and A∈Fin(X) and x∉A and P(A)
  shows P(cons(x,A))
```

```
proof -
  from prems have P(A∪{x}) by simp
  moreover have cons(x,A) = A∪{x} by auto
  ultimately show thesis by simp
qed
```

First order version of the induction for the finite powerset.

```
lemma Finite1_L16B: assumes A1: P(0) and A2: B∈Fin(X)
  and A3: ∀A∈Fin(X).∀x∈X. x∉A ∧ P(A)⟶P(A∪{x})
  shows P(B)
proof -
  from A1 have P(0) .
  with A2 show  P(B) using Finite1_L16A by (rule Fin_induct)
qed
```

8.2 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_x.thy series.

```
constdefs
  FinRangeFunctions(X,Y) ≡ {f:X→Y. f(X) ∈ Fin(Y)}
```

Constant functions have finite range.

```
lemma Finite1_L17: assumes c∈Y and X≠0
  shows ConstantFunction(X,c) ∈ FinRangeFunctions(X,Y)
  using prems  func1_3_L1 func_imagedef func1_3_L2 Finite1_L16
    FinRangeFunctions_def by simp
```

Finite range functions have finite range.

```
lemma Finite1_L18: assumes f ∈ FinRangeFunctions(X,Y)
  shows {f(x). x∈X} ∈ Fin(Y)
  using prems FinRangeFunctions_def func_imagedef by simp
```

An alternative form of the definition of finite range functions.

```
lemma Finite1_L19: assumes f:X→Y
  and {f(x). x∈X} ∈ Fin(Y)
  shows f ∈ FinRangeFunctions(X,Y)
  using prems func_imagedef FinRangeFunctions_def by simp
```

A composition of a finite range function with another function is a finite range function.

```
lemma Finite1_L20: assumes A1:f ∈ FinRangeFunctions(X,Y)
  and A2:g : Y→Z
  shows g O f ∈ FinRangeFunctions(X,Z)
proof -
  from A1 A2 have g{f(x). x∈X} ∈ Fin(Z)
```

```
    using Finite1_L18 Finite1_L6A
    by simp
  with A1 A2 have {(g O f)(x). x∈X} ∈ Fin(Z)
    using FinRangeFunctions_def apply_funtype
      func1_1_L17 comp_fun_apply by auto
  with A1 A2 show thesis using
    FinRangeFunctions_def comp_fun Finite1_L19
    by auto
qed
```

Image of any subset of the domain of a finite range function is finite.

```
lemma Finite1_L21:
  assumes A1: f ∈ FinRangeFunctions(X,Y) and A2: A⊆X
  shows f(A) ∈ Fin(Y)
proof -
  from A1 A2 have f(X) ∈ Fin(Y)  f(A) ⊆ f(X)
    using FinRangeFunctions_def func1_1_L8
    by auto
  then show f(A) ∈ Fin(Y) using Fin_subset_lemma
    by blast
qed

end
```

9 Finite_ZF.thy

```
theory Finite_ZF_1 imports Finite1 Order_ZF
```

begin

This theory file contains properties of finite sets related to order relations.

9.1 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 (cases A=0)
  assume A=0 then have T1: A∪{x} = {x} by simp
  from A1 have refl(X,r) using total_is_refl by simp
  with T1 A4 show A∪{x} = 0 ∨ HasAmaximum(r,A∪{x})
    using Order_ZF_4_L8 by simp
next assume A≠0
  with A1 A2 A3 A4 A5 show A∪{x} = 0 ∨ HasAmaximum(r,A∪{x})
    using FinD Order_ZF_4_L9 by simp
qed
```

For total and transitive relations finite set has a maximum.

```
theorem Finite_ZF_1_1_T1A:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: B∈Fin(X)
  shows B=0 ∨ HasAmaximum(r,B)
proof -
  have 0=0 ∨ HasAmaximum(r,0) by simp
  moreover from A3 have B∈Fin(X) .
  moreover from A1 A2 have ∀A∈Fin(X). ∀x∈X.
    x∉A ∧ (A=0 ∨ HasAmaximum(r,A)) ⟶ (A∪{x}=0 ∨ HasAmaximum(r,A∪{x}))
    using Finite_ZF_1_1_L1 by simp
  ultimately show  B=0 ∨ HasAmaximum(r,B) by (rule Finite1_L16B)
qed
```

Finite set has a minimum - induction step.

```
lemma Finite_ZF_1_1_L2:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: A∈Fin(X) and A4: x∈X and A5: A=0 ∨ HasAminimum(r,A)
  shows A∪{x} = 0 ∨ HasAminimum(r,A∪{x})
proof (cases A=0)
```

```
  assume A=0 then have T1: A∪{x} = {x} by simp
  from A1 have refl(X,r) using total_is_refl by simp
  with T1 A4 show A∪{x} = 0 ∨ HasAminimum(r,A∪{x})
    using Order_ZF_4_L8 by simp
next assume A≠0
  with A1 A2 A3 A4 A5 show A∪{x} = 0 ∨ HasAminimum(r,A∪{x})
    using FinD Order_ZF_4_L10 by simp
qed
```

For total and transitive relations finite set has a minimum.

```
theorem Finite_ZF_1_1_T1B:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: B ∈ Fin(X)
  shows B=0 ∨ HasAminimum(r,B)
proof -
  have 0=0 ∨ HasAminimum(r,0) by simp
  moreover from A3 have B∈Fin(X) .
  moreover from A1 A2 have ∀A∈Fin(X). ∀x∈X.
    x∉A ∧ (A=0 ∨ HasAminimum(r,A)) ⟶ (A∪{x}=0 ∨ HasAminimum(r,A∪{x}))
    using Finite_ZF_1_1_L2 by simp
  ultimately show  B=0 ∨ HasAminimum(r,B) by (rule Finite1_L16B)
qed
```

For transitive and total relations finite sets are bounded.

```
theorem Finite_ZF_1_T1:
  assumes A1: r {is total on} X and A2: trans(r)
  and A3: B∈Fin(X)
  shows IsBounded(B,r)
proof -
  from A1 A2 A3 have B=0 ∨ HasAminimum(r,B) B=0 ∨ HasAmaximum(r,B)
    using Finite_ZF_1_1_T1A Finite_ZF_1_1_T1B by auto
  then have
    B = 0 ∨ IsBoundedBelow(B,r) B = 0 ∨ IsBoundedAbove(B,r)
    using Order_ZF_4_L7 Order_ZF_4_L8A by auto
  then show IsBounded(B,r) using
    IsBounded_def IsBoundedBelow_def IsBoundedAbove_def
    by simp
qed
```

For linearly ordered finite sets maximum and minimum have desired properties. The reason we need linear order is that we need the order to be total and transitive for the finite sets to have a maximum and minimum and then we also need antisymmetry for the maximum and minimum to be unique.

```
theorem Finite_ZF_1_T2:
  assumes A1: IsLinOrder(X,r) and A2: A ∈ Fin(X) and A3: A≠0
  shows
  Maximum(r,A) ∈ A
  Minimum(r,A) ∈ A
```

```
  ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r
  ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
proof -
  from A1 have T1: r {is total on} X trans(r) antisym(r)
    using IsLinOrder_def by auto
  moreover from T1 A2 A3 have HasAmaximum(r,A)
    using Finite_ZF_1_1_T1A by auto
  moreover from T1 A2 A3 have HasAminimum(r,A)
    using Finite_ZF_1_1_T1B by auto
  ultimately show
    Maximum(r,A) ∈ A
    Minimum(r,A) ∈ A
    ∀x∈A. ⟨x,Maximum(r,A)⟩ ∈ r ∀x∈A. ⟨Minimum(r,A),x⟩ ∈ r
    using Order_ZF_4_L3 Order_ZF_4_L4 by auto
qed
```

A special case of `Finite_ZF_1_T2` when the set has three elements.

```
corollary Finite_ZF_1_L2A:
  assumes A1: IsLinOrder(X,r) and A2: a∈X  b∈X  c∈X
  shows
  Maximum(r,{a,b,c}) ∈ {a,b,c}
  Minimum(r,{a,b,c}) ∈ {a,b,c}
  Maximum(r,{a,b,c}) ∈ X
  Minimum(r,{a,b,c}) ∈ X
  ⟨a,Maximum(r,{a,b,c})⟩ ∈ r
  ⟨b,Maximum(r,{a,b,c})⟩ ∈ r
  ⟨c,Maximum(r,{a,b,c})⟩ ∈ r
proof -
  from A2 have I: {a,b,c} ∈ Fin(X)  {a,b,c} ≠ 0
    by auto
  with A1 show II: Maximum(r,{a,b,c}) ∈ {a,b,c}
    by (rule Finite_ZF_1_T2)
  moreover from A1 I show III: Minimum(r,{a,b,c}) ∈ {a,b,c}
    by (rule Finite_ZF_1_T2)
  moreover from A2 have {a,b,c} ⊆ X
    by auto
  ultimately show
    Maximum(r,{a,b,c}) ∈ X
    Minimum(r,{a,b,c}) ∈ X
    by auto
  from A1 I have ∀x∈{a,b,c}. ⟨x,Maximum(r,{a,b,c})⟩ ∈ r
    by (rule Finite_ZF_1_T2)
  then show
    ⟨a,Maximum(r,{a,b,c})⟩ ∈ r
    ⟨b,Maximum(r,{a,b,c})⟩ ∈ r
    ⟨c,Maximum(r,{a,b,c})⟩ ∈ r
    by auto
qed
```

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

can not be finite. Works for relations that are total, transitive and antisymmetric.

```
lemma Finite_ZF_1_1_L3:
  assumes A1: r {is total on} X
  and A2: trans(r) and A3: antisym(r)
  and A4: r ⊆ X×X and A5: X≠0
  and A6: ∀x∈X. ∃a∈A. x≠a ∧ ⟨x,a⟩ ∈ r
  shows A ∉ Fin(X)
proof -
  from prems have ¬IsBounded(A,r)
    using Order_ZF_3_L14 IsBounded_def
    by simp
  with A1 A2 show A ∉ Fin(X)
    using Finite_ZF_1_T1 by auto
qed

end
```

10 Topology_ZF.thy

theory `Topology_ZF` **imports** `Finite1 Fol1`

begin

This theory file provides basic definitions and properties of topology, open and closed sets, closure and boundary.

10.1 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. Hence, we decided to define a topology as a collection of sets that contains the empty set and 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 ∈`Pow(T)` then M is a subset of 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. Closed set is one such that it is contained in the carrier of the topology (i.e. $\bigcup T$) and its complement is open (i.e. belongs to the topology). The closure of a set is the intersection of all closed sets that contain it. To prove varius properties of closure we will often use the collection of closed sets that contain a given set A. Such collection does not have a name in romantic math. We will call it `ClosedCovers(A,T)`. The closure of a set A is defined as the intersection of the collection of the closed sets D such that $A \subseteq D$. We also define boundary of a set as the intersection of its closure with the closure of the complement (with respect to the carrier). 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 $\text{Fin}(M)$ is the collection of finite subsets of M (finite powerset of M), defined in the `Finite` theory of Isabelle/ZF.

constdefs

```
  IsATopology (_ {is a topology} [90] 91)
  T {is a topology} ≡ ( 0 ∈ T) ∧ ( ∀M∈Pow(T). ⋃M ∈ T ) ∧
  ( ∀U∈T. ∀ V∈T. U∩V ∈ T)

  Interior(A,T) ≡ ⋃ {U∈T. U⊆A}

  IsClosed (infixl {is closed in} 90)
  D {is closed in} T ≡ (D ⊆ ⋃T ∧ ⋃T - D ∈ T)

  ClosedCovers(A,T) ≡ {D ∈ Pow(⋃T). D {is closed in} T ∧ A⊆D}
```

```
Closure(A,T) ≡ ⋂ ClosedCovers(A,T)

Boundary(A,T) ≡ Closure(A,T) ∩ Closure(⋃T - A,T)


IsCompact (infixl {is compact in} 90)
K {is compact in} T ≡ (K ⊆ ⋃T ∧
(∀ M∈Pow(T). K ⊆ ⋃M ⟶ (∃ N∈Fin(M). K ⊆ ⋃N)))
```

A basic example of a topology: the powerset of any set is a topology.

```
lemma Top_1_L1: shows Pow(X) {is a topology}
proof -
  have 0 ∈ Pow(X) by simp
  moreover have ∀A∈Pow(Pow(X)). ⋃A ∈ Pow(X) by fast
  moreover have ∀U∈Pow(X). ∀V∈Pow(X). U∩V ∈ Pow(X) by fast
  ultimately show Pow(X) {is a topology} using IsATopology_def
    by auto
qed
```

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

```
lemma Top_1_L2: assumes A1: 𝓜 ≠ 0 and A2: ∀T∈𝓜. T {is a topology}
  shows (⋂𝓜) {is a topology}
proof -
  from A1 A2 have 0 ∈ ⋂𝓜 using IsATopology_def
    by auto
  moreover
  { fix A assume A∈Pow(⋂𝓜)
    with A1 have ∀T∈𝓜. A∈Pow(T) by auto
    with A1 A2 have ⋃A ∈ ⋂𝓜 using IsATopology_def
      by auto
  } then have ∀A. A∈Pow(⋂𝓜) ⟶ ⋃A ∈ ⋂𝓜 by simp
  hence ∀A∈Pow(⋂𝓜). ⋃A ∈ ⋂𝓜 by auto
  moreover
  { fix U V assume U ∈ ⋂𝓜 and V ∈ ⋂𝓜
    then have ∀T∈𝓜. U ∈ T ∧ V ∈ T by auto
    with A1 A2 have ∀T∈𝓜. U∩V ∈ T using IsATopology_def
      by simp
  } then have ∀ U ∈ ⋂𝓜. ∀ V ∈ ⋂𝓜. U∩V ∈ ⋂𝓜
    by auto
  ultimately show (⋂𝓜) {is a topology}
    using IsATopology_def by simp
qed
```

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 topolgy. The interior of the set A

(with respect to the topology in the context) is denoted int(A). The closure of a set $A \subseteq \bigcup T$ is denoted cl(A) and the boundary is ∂A.

```
locale topology0 =
  fixes T
  assumes topSpaceAssum: T {is a topology}

  fixes int
  defines int_def [simp]: int(A) ≡ Interior(A,T)

  fixes cl
  defines cl_def [simp]: cl(A) ≡ Closure(A,T)

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

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

```
lemma (in topology0) Top_1_L3: assumes N≠0 N ∈ Fin(T)
  shows ⋂N ∈ T
  using topSpaceAssum prems IsATopology_def Finite1_L5 by simp
```

Having a topology T and a set X we can define the induced topology as the one consisting of the intersections of X with sets from T. The notion of a collection restricted to a set is defined in Finite1.thy.

```
lemma (in topology0) Top_1_L4:
  shows (T {restricted to} X) {is a topology}
proof -
  let S = T {restricted to} X
  from topSpaceAssum have 0 ∈ S
    using IsATopology_def RestrictedTo_def by auto
  moreover have ∀A∈Pow(S). ⋃A ∈ S
  proof
    fix A assume A1: A∈Pow(S)
    from topSpaceAssum have ∀V∈A. ⋃ {U ∈ T. V = U∩X} ∈ T
      using IsATopology_def by auto
    hence {⋃{U∈T. V = U∩X}.V∈ A} ⊆ T by auto
    with topSpaceAssum have (⋃V∈A. ⋃{U∈T. V = U∩X}) ∈ T
      using IsATopology_def by auto
    then have (⋃V∈A. ⋃{U∈T. V = U∩X})∩ X ∈ S
      using RestrictedTo_def by auto
    moreover
    from A1 have ∀V∈A. ∃U∈T. V = U∩X
      using RestrictedTo_def by auto
    hence (⋃V∈A. ⋃{U∈T. V = U∩X})∩X = ⋃A by fast
    ultimately show ⋃A ∈ S by simp
  qed
  moreover have  ∀U∈S. ∀ V∈S. U∩V ∈ S
  proof -
    { fix U V assume U∈S  V∈S
```

```
    then obtain U₁ V₁ where
      U₁ ∈ T ∧ U = U₁∩X and V₁ ∈ T ∧ V = V₁∩X
      using RestrictedTo_def by auto
    with topSpaceAssum have U₁∩V₁ ∈ T and U∩V = (U₁∩V₁)∩X
      using IsATopology_def by auto
    then have  U∩V ∈ S using RestrictedTo_def by auto
  } then show ∀U∈S. ∀ V∈S. U∩V ∈ S
    by simp
  qed
  ultimately show S {is a topology} using IsATopology_def
    by simp
qed
```

10.2 Interior of a set

In section we show basic properties of the interior of a set.

Interior of a set A is contained in A.

```
lemma (in topology0) Top_2_L1: shows int(A) ⊆ A
  using Interior_def by auto
```

Interior is open.

```
lemma (in topology0) Top_2_L2: shows int(A) ∈ T
  using topSpaceAssum IsATopology_def Interior_def
  by auto
```

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

```
lemma (in topology0) Top_2_L3:  U∈T ⟷ int(U) = U
proof
  assume U∈T then show int(U) = U
    using Interior_def by auto
next assume A1: int(U) = U
  have int(U) ∈ T using Top_2_L2 by simp
  with A1 show U∈T by simp
qed
```

Interior of the interior is the interior.

```
lemma (in topology0) Top_2_L4: shows int(int(A)) = int(A)
proof -
  let U = int(A)
  from topSpaceAssum have U∈T using Top_2_L2 by simp
  then show int(int(A)) = int(A) using Top_2_L3 by simp
qed
```

Interior of a bigger set is bigger.

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

```
  from A1 have ∀ U∈T. (U⊆A ⟶ U⊆B) by auto
  then show int(A) ⊆ int(B) using Interior_def by auto
qed
```

An open subset of any set is a subset of the interior of that set.

```
lemma (in topology0) Top_2_L5: assumes U⊆A and U∈T
  shows U ⊆ int(A)
  using prems Interior_def by auto
```

If a point of a set has an open neighboorhood contained in the set, then the point belongs to the interior of the set.

```
lemma (in topology0) Top_2_L6:  assumes ∃U∈T. (x∈U ∧ U⊆A)
  shows x ∈ int(A)
  using prems Interior_def by auto
```

A set is open iff its every point has a an open neighbourhood contained in the set. We will formulate this statement as two lemmas (implication one way and the other way). The lemma below shows that if a set is open then every point has a an open neighbourhood contained in the set.

```
lemma (in topology0) Top_2_L7:
  assumes A1: V∈T
  shows ∀x∈V. ∃U∈T. (x∈U ∧ U⊆V)
proof -
  from A1 have ∀x∈V. V∈T ∧ x ∈ V ∧ V ⊆ V by simp
  then show thesis by auto
qed
```

If every point of a set has a an open neighbourhood contained in the set then the set is open.

```
lemma (in topology0) Top_2_L8:
  assumes A1: ∀x∈V. ∃U∈T. (x∈U ∧ U⊆V)
  shows V∈T
proof -
  from A1 have V = int(V) using Top_2_L1 Top_2_L6
    by blast
  then show V∈T using Top_2_L3 by simp
qed
```

10.3 Closed sets, closure, boundary.

This section is devoted to closed sets and properties of the closure and boundary operators.

The carrier of the space is closed.

```
lemma (in topology0) Top_3_L1: shows (⋃T) {is closed in} T
proof -
  have ⋃T - ⋃T = 0 by auto
```

```
  with topSpaceAssum have ⋃T - ⋃T ∈ T using IsATopology_def by auto
  then show thesis using IsClosed_def by simp
qed
```

Empty set is closed.

```
lemma (in topology0) Top_3_L2: shows 0 {is closed in} T
  using topSpaceAssum  IsATopology_def IsClosed_def by simp
```

The collection of closed covers of a subset of the carrier of topology is never empty. This is good to know, as we want to intersect this collection to get the closure.

```
lemma (in topology0) Top_3_L3:
  assumes A1: A ⊆ ⋃T shows ClosedCovers(A,T) ≠ 0
proof -
  from A1 have ⋃T ∈ ClosedCovers(A,T) using ClosedCovers_def Top_3_L1
    by auto
  then show thesis by auto
qed
```

Intersection of a nonempty family of closed sets is closed.

```
lemma (in topology0) Top_3_L4: assumes A1: K≠0 and
  A2: ∀D∈K. D {is closed in} T
  shows (⋂K) {is closed in} T
proof -
  from A2 have I: ∀D∈K. (D ⊆ ⋃T ∧ (⋃T - D)∈ T)
    using IsClosed_def by simp
  then have {⋃T - D. D∈ K} ⊆ T by auto
  with topSpaceAssum have (⋃ {⋃T - D. D∈ K}) ∈ T
    using IsATopology_def by auto
  moreover from A1 have ⋃ {⋃T - D. D∈ K} = ⋃T - ⋂K by fast
  moreover from A1 I have ⋂K ⊆ ⋃T by blast
  ultimately show (⋂K) {is closed in} T using  IsClosed_def
    by simp
qed
```

The union and intersection of two closed sets are closed.

```
lemma (in topology0) Top_3_L5:
  assumes A1: D₁ {is closed in} T   D₂ {is closed in} T
  shows
  (D₁∩D₂) {is closed in} T
  (D₁∪D₂) {is closed in} T
proof -
  have {D₁,D₂} ≠ 0 by simp
  with A1 have (⋂ {D₁,D₂}) {is closed in} T using Top_3_L4
    by fast
  thus (D₁∩D₂) {is closed in} T by simp
  from topSpaceAssum A1 have (⋃T - D₁) ∩ (⋃T - D₂) ∈ T
    using IsClosed_def IsATopology_def by simp
```

```
  moreover have (⋃T - D₁) ∩ (⋃T - D₂) = ⋃T - (D₁ ∪ D₂)
    by auto
  moreover from A1 have D₁ ∪ D₂ ⊆ ⋃T using IsClosed_def
    by auto
  ultimately show (D₁∪D₂) {is closed in} T using IsClosed_def
    by simp
qed
```

Finite union of closed sets is closed. To understand the proof recall that D ∈Pow(⋃T) means that D is as subset of the carrier of the topology.

```
lemma (in topology0) Top_3_L6:
  assumes A1: N ∈ Fin({D∈Pow(⋃T). D {is closed in} T})
  shows (⋃N) {is closed in} T
proof -
  let C = {D∈Pow(⋃T). D {is closed in} T}
  have 0∈C using Top_3_L2 by simp
  moreover have ∀A B. ((A∈C ∧  B∈C) ⟶ A∪B ∈ C)
    using Top_3_L5 by auto
  ultimately have ⋃N ∈ C by (rule Finite1_L3)
  thus (⋃N) {is closed in} T by simp
qed
```

Closure of a set is closed.

```
lemma (in topology0) Top_3_L7: assumes A ⊆ ⋃T
  shows cl(A) {is closed in} T
  using prems Closure_def Top_3_L3 ClosedCovers_def Top_3_L4
  by simp
```

Closure of a bigger sets is bigger.

```
lemma (in topology0) top_closure_mono:
  assumes A1: A ⊆ ⋃T  B ⊆ ⋃T  and A2:A⊆B
  shows cl(A) ⊆ cl(B)
proof -
  from A2 have ClosedCovers(B,T)⊆ ClosedCovers(A,T)
    using ClosedCovers_def by auto
  with A1 show thesis using Top_3_L3 Closure_def by auto
qed
```

Boundary of a set is closed.

```
lemma (in topology0) boundary_closed:
  assumes A1: A ⊆ ⋃T shows ∂A {is closed in} T
proof -
  from A1 have ⋃T - A ⊆ ⋃T by fast
  with A1 show ∂A {is closed in} T
    using Top_3_L7 Top_3_L5 Boundary_def by auto
qed
```

A set is closed iff it is equal to its closure.

```
lemma (in topology0) Top_3_L8: assumes A1: A ⊆ ⋃T
  shows A {is closed in} T ⟷ cl(A) = A
proof
  assume A {is closed in} T
  with A1 show cl(A) = A
    using Closure_def ClosedCovers_def by auto
next assume cl(A) = A
  then have ⋃T - A = ⋃T - cl(A) by simp
  with A1 show A {is closed in} T using Top_3_L7 IsClosed_def
    by simp
qed
```

Complement of an open set is closed.

```
lemma (in topology0) Top_3_L9:
  assumes A1: A∈T
  shows (⋃T - A) {is closed in} T
proof -
  from topSpaceAssum A1 have ⋃T - (⋃T - A) = A and ⋃T - A ⊆ ⋃T
    using IsATopology_def by auto
  with A1 show (⋃T - A) {is closed in} T using IsClosed_def by simp
qed
```

A set is contained in its closure.

```
lemma (in topology0) Top_3_L10: assumes A ⊆ ⋃T shows A ⊆ cl(A)
  using prems Top_3_L1 ClosedCovers_def Top_3_L3 Closure_def by auto
```

Closure of a subset of the carrier is a subset of the carrier and closure of the complement is the complement of the interior.

```
lemma (in topology0) Top_3_L11: assumes A1: A ⊆ ⋃T
  shows
  cl(A) ⊆ ⋃T
  cl(⋃T - A) = ⋃T - int(A)
proof -
  from A1 show cl(A) ⊆ ⋃T using Top_3_L1 Closure_def ClosedCovers_def
    by auto
  from A1 have ⋃T - A ⊆ ⋃T - int(A) using Top_2_L1
    by auto
  moreover have I: ⋃T - int(A) ⊆ ⋃T   ⋃T - A ⊆ ⋃T by auto
  ultimately have cl(⋃T - A) ⊆ cl(⋃T - int(A))
    using top_closure_mono by simp
  moreover
  from I have (⋃T - int(A)) {is closed in} T
    using Top_2_L2 Top_3_L9 by simp
  with I have cl((⋃T) - int(A)) = ⋃T - int(A)
    using Top_3_L8 by simp
  ultimately have cl(⋃T - A) ⊆ ⋃T - int(A) by simp
  moreover
  from I have ⋃T - A ⊆ cl(⋃T - A) using Top_3_L10 by simp
  hence ⋃T - cl(⋃T - A) ⊆ A and ⋃T - A ⊆ ⋃T  by auto
```

```
  then have ⋃T - cl(⋃T - A) ⊆ int(A)
    using Top_3_L7 IsClosed_def Top_2_L5 by simp
  hence ⋃T - int(A) ⊆  cl(⋃T - A) by auto
  ultimately show cl(⋃T - A) = ⋃T - int(A) by auto
qed
```

Boundary of a set is the closure of the set minus the interior of the set.

```
lemma (in topology0) Top_3_L12: assumes A1: A ⊆ ⋃T
  shows ∂A = cl(A) - int(A)
proof -
  from A1 have ∂A = cl(A) ∩ (⋃T - int(A))
    using Boundary_def Top_3_L11 by simp
  moreover from A1 have
    cl(A) ∩ (⋃T - int(A)) = cl(A) - int(A)
    using Top_3_L11 by blast
  ultimately show ∂A = cl(A) - int(A) by simp
qed
```

If a set A is contained in a closed set B, then the closure of A is contained in B.

```
lemma (in topology0) Top_3_L13:
  assumes A1: B {is closed in} T   A⊆B
  shows cl(A) ⊆ B
proof -
  from A1 have B ⊆ ⋃T using IsClosed_def by simp
  with A1 show cl(A) ⊆ B using ClosedCovers_def Closure_def by auto
qed
```

If two open sets are disjoint, then we can close one of them and they will still be disjoint.

```
lemma (in topology0) Top_3_L14:
  assumes A1: U∈T  V∈T and  A2: U∩V = 0
  shows cl(U) ∩ V = 0
proof -
  from topSpaceAssum A1 have I: U ⊆ ⋃T using IsATopology_def
    by auto
  with A2 have  U ⊆ ⋃T - V by auto
  moreover from A1 have (⋃T - V) {is closed in} T using Top_3_L9
    by simp
  ultimately have cl(U) - (⋃T - V) = 0
    using Top_3_L13 by blast
  moreover
  from I have cl(U) ⊆ ⋃T using Top_3_L7 IsClosed_def by simp
  then have cl(U) -(⋃T - V) = cl(U) ∩ V by auto
  ultimately show cl(U) ∩ V = 0 by simp
qed

end
```

11 Topology_ZF_1.thy

theory Topology_ZF_1 **imports** Topology_ZF Fol1

begin

11.1 Separation axioms.

Topological spaces cas be classified according to certain properties called "separation axioms". This section defines 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. A topology is T_1 if for every such pair there exist an open set that contains the first point but not the second. 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.

constdefs

```
isT0 (_ {is T₀} [90] 91)
T {is T₀} ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
(∃U∈T. (x∈U ∧ y∉U) ∨ (y∈U ∧ x∉U)))

isT1 (_ {is T₁} [90] 91)
T {is T₁} ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
(∃U∈T. (x∈U ∧ y∉U)))

isT2 (_ {is T₂} [90] 91)
T {is T₂} ≡ ∀ x y. ((x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y) ⟶
(∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0))
```

If a topology is T_1 then it is T_0. We don't really assume here that T is a topology on X. Instead, we prove the relation between isT0 condition and isT1.

```
lemma T1_is_T0: assumes A1: T {is T₁} shows T {is T₀}
proof -
  from A1 have ∀ x y. x ∈ ⋃T ∧ y ∈ ⋃T ∧ x≠y ⟶
    (∃U∈T. x∈U ∧ y∉U)
    using isT1_def by simp
  then have ∀ x y. x ∈ ⋃T ∧ y ∈ ⋃T ∧ x≠y ⟶
    (∃U∈T. x∈U ∧ y∉U ∨ y∈U ∧ x∉U)
    by auto
  then show T {is T₀} using isT0_def by simp
qed
```

If a topology is T_2 then it is T_1.

```
lemma T2_is_T1: assumes A1: T {is T₂} shows T {is T₁}
proof -
```

```
  { fix x y assume x ∈ ⋃T  y ∈ ⋃T  x≠y
    with A1 have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0
      using isT2_def by auto
    then have ∃U∈T. x∈U ∧ y∉U by auto
  } then have ∀ x y. x ∈ ⋃T ∧ y ∈ ⋃T ∧  x≠y ⟶
      (∃U∈T. x∈U ∧ y∉U) by simp
  then show T {is T₁} using isT1_def by simp
qed
```

In a T_0 space two points that can not be separated by an open set are equal. Proof by contradiction.

```
lemma Top_1_1_L1: assumes A1: T {is T₀} and A2: x ∈ ⋃T  y ∈ ⋃T
  and A3: ∀U∈T. (x∈U ⟷ y∈U)
  shows x=y
proof -
  { assume x≠y
    with A1 A2 have ∃U∈T. x∈U ∧ y∉U ∨ y∈U ∧ x∉U
      using isT0_def by simp
    with A3 have False by auto
  } then show x=y by auto
qed
```

In a T_2 space two points can be separated by an open set with its boundary.

```
lemma (in topology0) Top_1_1_L2:
  assumes A1: T {is T₂}  and A2: x ∈ ⋃T  y ∈ ⋃T   x≠y
  shows ∃U∈T. (x∈U ∧ y ∉ cl(U))
proof -
  from A1 A2 have ∃U∈T. ∃V∈T. x∈U ∧ y∈V ∧ U∩V=0
    using isT2_def by simp
  then obtain U V where U∈T  V∈T  x∈U  y∈V  U∩V=0
    by auto
  then have U∈T ∧ x∈U ∧ y∈ V ∧ cl(U) ∩ V = 0 using Top_3_L14
    by simp
  then show ∃U∈T. (x∈U ∧ y ∉ cl(U)) by auto
qed
```

In a T_2 space compact sets are closed. Doing a formal proof of this theorem gave me an interesting insight into the role of the Axiom of Choice in romantic proofs.

A typical romantic 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 have never realized that such reasoning requires (an) 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 Top_1_1_L2 above). This only states that the set of such open sets U is not empty. To get the collection $\{U_x\}_{x\in K}$ in the above proof we have to select one such set among many for every $x \in K$ and this is where we use (an) Axiom of Choice. Probably in 99/100 cases when a romatic calculus proof states something like $\forall \varepsilon \exists \delta_\varepsilon \cdots$ the proof uses Axiom of Choice. In the proof below we avoid using Axiom of Choice (read it to find out how). It is an interesting question which such calculus proofs can be reformulated so that the usage of AC is avoided. I remember Sierpiński published a paper in 1919 (or was it 1914? my memory is not that good any more) where he showed that one needs an Axiom of Choice to show the equivalence of the Heine and Cauchy definitions of limits.

```
theorem (in topology0) in_t2_compact_is_cl:
  assumes A1: T {is T₂} and A2: K {is compact in} T
  shows K {is closed in} T
proof -
  { fix y assume A3: y ∈ ⋃T  y∉K
    have ∃U∈T. y∈U ∧ U ⊆ ⋃T - K
    proof -
      let B = ⋃x∈K.{V∈T. x∈V ∧ y∉ cl(V)}
      have I: B ∈ Pow(T)  Fin(B) ⊆ Pow(B)
        using Fin.dom_subset by auto
      from A2 A3 have ∀x∈K. x ∈ ⋃T ∧ y ∈ ⋃T ∧ x≠y
        using IsCompact_def by auto
      with A1 have ∀x∈K. {V∈T. x∈V ∧ y ∉ cl(V)} ≠ 0
        using Top_1_1_L2 by auto
      hence K ⊆ ⋃B by blast
      with A2 I have ∃N ∈ Fin(B). K ⊆ ⋃N using IsCompact_def
        by auto
      then obtain N where D1: N ∈ Fin(B)  K ⊆ ⋃N
        by auto
      with I have N ⊆ B by auto
      hence II: ∀V∈N. V∈B by auto
      let M = {cl(V). V∈N}
      let C = {D∈Pow(⋃T). D {is closed in} T}
      from topSpaceAssum have
        ∀V∈B. (cl(V) {is closed in} T)
        ∀V∈B. (cl(V) ∈ Pow(⋃T))
        using IsATopology_def Top_3_L7 IsClosed_def
        by auto
      hence ∀V∈B. cl(V) ∈ C by simp
      moreover from D1 have N ∈ Fin(B) by simp
      ultimately have M ∈ Fin(C) by (rule Finite1_L6)
      then have ⋃T - ⋃M ∈ T using Top_3_L6 IsClosed_def
        by simp
      moreover from A3 II have y ∈ ⋃T - ⋃M by simp
      moreover have ⋃T - ⋃M ⊆ ⋃T - K
      proof -
        from II have ⋃N ⊆ ⋃M using Top_3_L10 by auto
        with D1 show ⋃T - ⋃M ⊆ ⋃T - K by auto
      qed
```

```
      ultimately have ∃U. U∈T ∧ y ∈ U ∧ U ⊆ ⋃T - K
        by auto
      then show ∃U∈T. y∈U ∧ U ⊆ ⋃T - K by auto
    qed
  } then have ∀y ∈ ⋃T - K. ∃U∈T. y∈U ∧ U ⊆ ⋃T - K
    by auto
  with A2 show K {is closed in} T
    using Top_2_L8 IsCompact_def IsClosed_def by auto
qed
```

11.2 Bases and subbases.

A base of topology is a collection of open sets such that every open set is a union of the sets from the base. A subbase is a collection of open sets such that finite intersection of those sets form a base. 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 $U \cap V$.

constdefs

```
  IsAbaseFor (infixl {is a base for} 65)
  B {is a base for} T ≡ B⊆T ∧ T = {⋃A. A∈Pow(B)}

  IsAsubBaseFor (infixl {is a subbase for} 65)
  B {is a subbase for} T ≡
  B ⊆ T ∧ {⋂A. A∈Fin(B)} {is a base for} T

  SatisfiesBaseCondition (_ {satisfies the base condition} [50] 50)
  B {satisfies the base condition} ≡
  ∀U V. ((U∈B ∧ V∈B) ⟶ (∀x ∈ U∩V. ∃W∈B. x∈W ∧ W ⊆ U∩V))
```

Each open set is a union of some sets from the base.

```
lemma Top_1_2_L1: assumes B {is a base for} T  and U∈T
  shows ∃A∈Pow(B). U = ⋃A
  using prems IsAbaseFor_def by simp
```

A necessary conditionfor a collection of sets to be a base for some topology : every point in the intersection of two sets in the base has a neighboorhood from the base contained in the intersection.

```
lemma Top_1_2_L2:
  assumes A1:∃T. T {is a topology} ∧ B {is a base for} T
  and A2: V∈B  W∈B
  shows ∀ x ∈ V∩W. ∃U∈B. x∈U ∧ U ⊆ V ∩ W
proof -
  from A1 obtain T where
    D1: T {is a topology}   B {is a base for} T
```

```
    by auto
  then have B ⊆ T using IsAbaseFor_def by auto
  with A2 have V∈T and W∈T using IsAbaseFor_def by auto
  with D1 have ∃A∈Pow(B). V∩W = ⋃A using IsATopology_def Top_1_2_L1
    by auto
  then obtain A where A ⊆ B and V ∩ W = ⋃A by auto
  then show ∀ x ∈ V∩W. ∃U∈B. (x∈U ∧ U ⊆ V ∩ W) by auto
qed
```

We will construct a topology as the collection of unions of (would-be) base. First we prove that if the collection of sets satisfies the condition we want to show to be sufficient, the the intersection belongs to what we will define as topology (am I clear here?). Having this fact ready simplifies the proof of the next lemma. There is not much topology here, just some set theory.

```
lemma Top_1_2_L3:
  assumes A1: ∀x∈ V∩W . ∃U∈B. x∈U ∧ U ⊆ V∩W
  shows V∩W ∈ {⋃A. A∈Pow(B)}
proof
  let A = ⋃x∈V∩W. {U∈B. x∈U ∧ U ⊆ V∩W}
  show A∈Pow(B) by auto
  from A1 show V∩W = ⋃A by blast
qed
```

The next lemma is needed when proving that the would-be topology is closed with respect to taking intersections. We show here that intersection of two sets from this (would-be) topology can be written as union of sets from the topology.

```
lemma Top_1_2_L4:
  assumes A1: U1 ∈ {⋃A. A∈Pow(B)}  U2 ∈ {⋃A. A∈Pow(B)}
  and A2: B {satisfies the base condition}
  shows ∃C. C ⊆ {⋃A. A∈Pow(B)} ∧ U1∩U2 = ⋃C
proof -
  from A1 A2 obtain A1 A2 where
    D1: A1∈ Pow(B)  U1 = ⋃A1  A2 ∈ Pow(B)  U2 = ⋃A2
    by auto
  let C = ⋃U∈A1.{U∩V. V∈A2}
  from D1 have (∀U∈A1. U∈B) ∧ (∀V∈A2. V∈B) by auto
  with A2 have C ⊆ {⋃A . A ∈ Pow(B)}
    using Top_1_2_L3 SatisfiesBaseCondition_def by auto
  moreover from D1 have U1 ∩ U2 = ⋃C by auto
  ultimately show thesis by auto
qed
```

If B satisfies the base condition, then the collection of unions of sets from B is a topology and B is a base for this topology.

```
theorem Top_1_2_T1:
  assumes A1: B {satisfies the base condition}
  and A2: T = {⋃A. A∈Pow(B)}
```

```
  shows T {is a topology} and B {is a base for} T
proof -
  show T {is a topology}
  proof -
    from A2 have 0∈T by auto
    moreover have I: ∀C∈Pow(T). ⋃C ∈ T
    proof -
      { fix C assume A3: C ∈ Pow(T)
        let Q = ⋃ {⋃{A∈Pow(B). U = ⋃A}. U∈C}
        from A2 A3 have ∀U∈C. ∃A∈Pow(B). U = ⋃A by auto
        then have ⋃Q = ⋃C using  Finite1_L8 by simp
        moreover from A2 have ⋃Q ∈ T by auto
        ultimately have ⋃C ∈ T by simp
      } thus ∀C∈Pow(T). ⋃C ∈ T by auto
    qed
    moreover have ∀U∈T. ∀ V∈T. U∩V ∈ T
    proof -
      { fix U V assume  U ∈ T  V ∈ T
        with A1 A2 have ∃C.(C ⊆ T ∧ U∩V = ⋃C)
          using Top_1_2_L4 by simp
        then obtain C where C ⊆ T and  U∩V = ⋃C
          by auto
        with I have U∩V ∈ T by simp
      } then show ∀U∈T. ∀ V∈T. U∩V ∈ T by simp
    qed
    ultimately show T {is a topology} using IsATopology_def
      by simp
  qed
  from A2 have B⊆T by auto
  with A2 show B {is a base for} T using IsAbaseFor_def
    by simp
qed
```

The carrier of the base and topology are the same.

```
lemma Top_1_2_L5: assumes B {is a base for} T
  shows ⋃T = ⋃B
  using prems IsAbaseFor_def by auto
```

11.3 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 $A \in S, B \in T$.

constdefs

```
  ProductTopology(T,S) ≡ {⋃W. W ∈ Pow(ProductCollection(T,S))}
```

The product collection satisfies the base condition.

```
lemma Top_1_4_L1:
  assumes A1: T {is a topology}   S {is a topology}
  and A2: A ∈ ProductCollection(T,S)  B ∈ ProductCollection(T,S)
  shows ∀x∈(A∩B). ∃W∈ProductCollection(T,S). (x∈W ∧ W ⊆ A ∩ B)
proof
  fix x assume A3: x ∈ A∩B
  from A2 obtain U₁ V₁ U₂ V₂ where
    D1: U₁∈T  V₁∈S   A=U₁×V₁  U₂∈T  V₂∈S   B=U₂×V₂
    using ProductCollection_def by auto
  let W = (U₁∩U₂) × (V₁∩V₂)
  from A1 D1 have U₁∩U₂ ∈ T and V₁∩V₂ ∈ S
    using IsATopology_def by auto
  then have W ∈ ProductCollection(T,S) using ProductCollection_def
    by auto
  moreover from A3 D1 have x∈W and W ⊆ A∩B by auto
  ultimately have ∃W. (W ∈ ProductCollection(T,S) ∧ x∈W ∧ W ⊆ A∩B)
    by auto
  thus ∃W∈ProductCollection(T,S). (x∈W ∧ W ⊆ A ∩ B) by auto
qed
```

The product topology is indeed a topology on the product.

```
theorem Top_1_4_T1: assumes A1: T {is a topology}  S {is a topology}
  shows
  ProductTopology(T,S) {is a topology}
  ProductCollection(T,S) {is a base for} ProductTopology(T,S)
  ⋃ ProductTopology(T,S) = ⋃T × ⋃S
proof -
  from A1 show
    ProductTopology(T,S) {is a topology}
    ProductCollection(T,S) {is a base for} ProductTopology(T,S)
    using Top_1_4_L1 ProductCollection_def
      SatisfiesBaseCondition_def ProductTopology_def Top_1_2_T1
    by auto
  then show ⋃ ProductTopology(T,S) = ⋃T × ⋃S
    using Top_1_2_L5 ZF1_1_L6 by simp
qed

end
```

12 Topology_ZF_2.thy

```
theory Topology_ZF_2 imports Topology_ZF_1 func1 Fol1

begin
```

12.1 Continuous functions.

In standard math we say that a function is contiuous 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.

Recall that in Isabelle/ZF `f-(A)` denotes the inverse image of (set) A with respect to (function) f.

```
constdefs
  IsContinuous(τ1,τ2,f) ≡ (∀U∈τ2. f-(U) ∈ τ1)
```

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 function $f : X_1 \to X_2$, where X_i is defined as $\bigcup \tau_i$ for $i = 1, 2$. We also define notation $\mathtt{cl}_1\mathtt{(A)}$ and $\mathtt{cl}_2\mathtt{(A)}$ for closure of a set A in topologies τ_1 and τ_2, respectively.

```
locale two_top_spaces0 =

  fixes τ1
  assumes tau1_is_top: τ1 {is a topology}

  fixes τ2
  assumes tau2_is_top: τ2 {is a topology}

  fixes X1
  defines X1_def [simp]: X1 ≡ ⋃τ1

  fixes X2
  defines X2_def [simp]: X2 ≡ ⋃τ2


  fixes f
  assumes fmapAssum: f: X1 → X2

  fixes isContinuous (_ {is continuous} [50] 50)
  defines isContinuous_def [simp]: g {is continuous} ≡ IsContinuous(τ1,τ2,g)

  fixes cl1
  defines cl1_def [simp]: cl1(A) ≡ Closure(A,τ1)
```

```
  fixes cl₂
  defines cl₂_def [simp]: cl₂(A) ≡ Closure(A,τ₂)
```

First we show that theorems proven in locale topology0 are valid when applied to topologies τ_1 and τ_2.

```
lemma (in two_top_spaces0) topol_cntxs_valid:
  shows topology0(τ₁) and topology0(τ₂)
  using tau1_is_top tau2_is_top topology0_def by auto
```

For continuous functions the inverse image of a closed set is closed.

```
lemma (in two_top_spaces0) TopZF_2_1_L1:
  assumes A1: f {is continuous} and A2: D {is closed in} τ₂
  shows f-(D) {is closed in} τ₁
proof -
  from fmapAssum have  f-(D) ⊆ X₁ using func1_1_L3 by simp
  moreover from fmapAssum have f-(X₂ - D) =  X₁ - f-(D)
    using Pi_iff function_vimage_Diff func1_1_L4 by auto
  ultimately have X₁ - f-(X₂ - D) = f-(D) by auto
  moreover from A1 A2 have (X₁ - f-(X₂ - D)) {is closed in} τ₁
    using IsClosed_def IsContinuous_def topol_cntxs_valid topology0.Top_3_L9
    by simp
  ultimately show f-(D) {is closed in} τ₁ by simp
qed
```

If the inverse image of every closed set is closed, then the image of a closure is contained in the closure of the image.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L2:
  assumes A1: ∀D. ((D {is closed in} τ₂) ⟶ f-(D) {is closed in} τ₁)
  and A2: A ⊆ X₁
  shows f(cl₁(A)) ⊆ cl₂(f(A))
proof -
  from fmapAssum have f(A) ⊆ cl₂(f(A))
    using func1_1_L6 topol_cntxs_valid topology0.Top_3_L10
    by simp
  with fmapAssum have f-(f(A)) ⊆ f-(cl₂(f(A)))
    using func1_1_L7 by auto
  moreover from fmapAssum A2 have A ⊆ f-(f(A))
    using func1_1_L9 by simp
  ultimately have A ⊆ f-(cl₂(f(A))) by auto
  with fmapAssum A1 have f(cl₁(A)) ⊆ f(f-(cl₂(f(A))))
    using func1_1_L6 func1_1_L8 IsClosed_def
      topol_cntxs_valid topology0.Top_3_L7 topology0.Top_3_L13
    by simp
  moreover from fmapAssum have f(f-(cl₂(f(A)))) ⊆ cl₂(f(A))
    using fun_is_function function_image_vimage by simp
  ultimately show f(cl₁(A)) ⊆ cl₂(f(A))
    by auto
qed
```

If $f\left(\overline{A}\right) \subseteq \overline{f(A)}$ (the image of the closure is contained in the closure of the image), then $\overline{f^{-1}(B)} \subseteq f^{-1}\left(\overline{B}\right)$ (the inverse image of the closure contains the closure of the inverse image).

```
lemma (in two_top_spaces0) Top_ZF_2_1_L3:
  assumes A1: ∀ A. ( A ⊆ X₁ ⟶ f(cl₁(A)) ⊆ cl₂(f(A)))
  shows ∀B. ( B ⊆ X₂ ⟶ cl₁(f-(B)) ⊆ f-(cl₂(B)) )
proof -
  { fix B assume A2: B ⊆ X₂
    from fmapAssum A1 have f(cl₁(f-(B))) ⊆ cl₂(f(f-(B)))
      using func1_1_L3 by simp
    moreover from fmapAssum A2 have cl₂(f(f-(B))) ⊆ cl₂(B)
      using fun_is_function function_image_vimage func1_1_L6
        topol_cntxs_valid topology0.top_closure_mono
      by simp
    ultimately have f-(f(cl₁(f-(B)))) ⊆ f-(cl₂(B))
      using fmapAssum fun_is_function func1_1_L7 by auto
    moreover from fmapAssum A2 have
      cl₁(f-(B)) ⊆ f-(f(cl₁(f-(B))))
      using func1_1_L3 func1_1_L9 IsClosed_def
        topol_cntxs_valid topology0.Top_3_L7 by simp
    ultimately have cl₁(f-(B)) ⊆ f-(cl₂(B)) by auto
  } then show thesis by simp
qed
```

If $\overline{f^{-1}(B)} \subseteq f^{-1}\left(\overline{B}\right)$ (the inverse image of a closure contains the closure of the inverse image), then the function is continuous. This lemma closes a series of implications showing equavalence of four definitions of continuity.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L4:
  assumes A1: ∀B. ( B ⊆ X₂ ⟶ cl₁(f-(B)) ⊆ f-(cl₂(B)) )
  shows f {is continuous}
proof -
  { fix U assume A2: U ∈ τ₂
    from A2 have (X₂ - U) {is closed in} τ₂
      using topol_cntxs_valid topology0.Top_3_L9 by simp
    moreover have X₂ - U ⊆ ⋃τ₂ by auto
    ultimately have cl₂(X₂ - U) = X₂ - U
      using topol_cntxs_valid topology0.Top_3_L8 by simp
    moreover from A1 have cl₁(f-(X₂ - U)) ⊆ f-(cl₂(X₂ - U))
      by auto
    ultimately have cl₁(f-(X₂ - U)) ⊆ f-(X₂ - U) by simp
    moreover from fmapAssum have f-(X₂ - U) ⊆ cl₁(f-(X₂ - U))
      using func1_1_L3 topol_cntxs_valid topology0.Top_3_L10
      by simp
    ultimately have f-(X₂ - U) {is closed in} τ₁
      using fmapAssum func1_1_L3 topol_cntxs_valid topology0.Top_3_L8
      by auto
    with fmapAssum have f-(U) ∈ τ₁
      using fun_is_function function_vimage_Diff func1_1_L4
```

```
      func1_1_L3 IsClosed_def double_complement by simp
  } then have ∀U∈τ₂. f-(U) ∈ τ₁ by simp
  then show thesis using IsContinuous_def by simp
qed
```

Another condition for continuity: it is sufficient to check if the inverse image of every set in a base is open.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L5:
  assumes A1: B {is a base for} τ₂ and A2: ∀U∈B. f-(U) ∈ τ₁
  shows f {is continuous}
proof -
  { fix V assume A3: V ∈ τ₂
    with A1 obtain A where D1: A ⊆ B  V = ⋃A
      using IsAbaseFor_def by auto
    with A2 have {f-(U). U∈A} ⊆ τ₁ by auto
    with tau1_is_top have ⋃ {f-(U). U∈A} ∈ τ₁
      using IsATopology_def by simp
    moreover from D1 have f-(V) = ⋃{f-(U). U∈A} by auto
    ultimately have f-(V) ∈  τ₁ by simp
  } then show f {is continuous} using IsContinuous_def
    by simp
qed
```

We can strenghten the previous lemma: it is sufficient to check if the inverse image of every set in a subbase is open. The proof is rather awkward, as usual when we deal with general intersections. We have to keep track of the case when the collection is empty.

```
lemma (in two_top_spaces0) Top_ZF_2_1_L6:
  assumes A1: B {is a subbase for} τ₂ and A2: ∀U∈B. f-(U) ∈ τ₁
  shows f {is continuous}
proof -
  let C = {⋂A. A ∈ Fin(B)}
  from A1 have C {is a base for} τ₂
    using IsAsubBaseFor_def by simp
  moreover have ∀U∈C. f-(U) ∈ τ₁
  proof
    fix U assume A3: U∈C
    { assume f-(U)=0
      with tau1_is_top have f-(U) ∈ τ₁
        using IsATopology_def by simp}
    moreover
    { assume A4: f-(U)≠0
      then have U≠0 by (rule func1_1_L13)
      moreover from A3 obtain A where
        D1:A ∈ Fin(B) and D2: U = ⋂A
        by auto
      ultimately have ⋂A≠0 by simp
      hence I: A≠0 by (rule Finite1_L9)
      then have {f-(W). W∈A} ≠ 0 by simp
```

```
      moreover from A2 D1 have {f-(W). W∈A} ∈ Fin(τ₁)
        by (rule Finite1_L6)
      ultimately have ⋂{f-(W). W∈A} ∈ τ₁
        using topol_cntxs_valid topology0.Top_1_L3 by simp
      moreover
      from A1 D1 have A ⊆ τ₂
        using FinD IsAsubBaseFor_def by auto
      with tau2_is_top have A ⊆ Pow(X₂)
        using IsATopology_def by auto
      with fmapAssum I have f-(⋂A) =  ⋂{f-(W). W∈A}
        using func1_1_L12 by simp
      with D2 have f-(U) = ⋂{f-(W). W∈A}
        by simp
      ultimately have f-(U) ∈ τ₁ by simp }
    ultimately show f-(U) ∈ τ₁ by blast
  qed
  ultimately show f {is continuous}
    using Top_ZF_2_1_L5 by simp
qed


end
```

13 Group_ZF.thy

theory Group_ZF **imports** func_ZF

begin

This theory file will cover basics of group theory.

13.1 Monoids.

Monoid is a set with an associative operation and a neutral element. The operation is of course a function on $G \times G$ with values in G, and therefore it is a subset of $(G \times G) \times G$. Those who don't like that can go to HOL. Monoid is like a group except that we don't require existence of the inverse.

```
constdefs
  IsAmonoid(G,f) ≡
  f {is associative on} G ∧
  (∃e∈G. (∀ g∈G. ( (f(<e,g>) = g) ∧ (f(<g,e>) = g))))
```

We use locales to define notation. This allows to separate notation and notion definitions. We would like to use additive notation for monoid, but unfortunately + is already taken.

```
locale monoid0 =
  fixes G and f
  assumes monoidAsssum:IsAmonoid(G,f)

  fixes monoper (infixl ⊕ 70)
  defines monoper_def [simp]: a ⊕ b ≡ f<a,b>
```

The result of the monoid operation is in the monoid (carrier).

```
lemma (in monoid0) group0_1_L1:
  assumes a∈G  b∈G shows a⊕b ∈ G
  using prems monoidAsssum IsAmonoid_def IsAssociative_def apply_funtype
  by auto
```

There is only one neutral element in monoid.

```
lemma (in monoid0) group0_1_L2:
  ∃!e. e∈G ∧ (∀ g∈G. ( (e⊕g = g) ∧ g⊕e = g))
proof
  fix e y
  assume e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
    and y ∈ G ∧ (∀g∈G. y ⊕ g = g ∧ g ⊕ y = g)
  then have y⊕e = y y⊕e = e by auto
  thus e = y by simp
next from monoidAsssum show
    ∃e. e∈ G ∧ (∀ g∈G. e⊕g = g ∧ g⊕e = g)
    using IsAmonoid_def by auto
```

qed

We could put the definition of neutral element anywhere, but it is only usable in conjuction with the above lemma.

```
constdefs
 TheNeutralElement(G,f) ≡
  ( THE e. e∈G ∧ (∀ g∈G. f<e,g> = g ∧ f<g,e> = g))
```

The neutral element is neutral.

```
lemma (in monoid0) group0_1_L3:
  assumes A1: e = TheNeutralElement(G,f)
  shows e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
proof -
  let n =  THE b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
  have ∃!b. b∈ G ∧ (∀ g∈G. b⊕g = g ∧ g⊕b = g)
    using group0_1_L2 by simp
  hence n∈ G ∧ (∀ g∈G. n⊕g = g ∧ g⊕n = g)
    by (rule theI)
  with A1 show thesis
    using TheNeutralElement_def by simp
qed
```

The monoid carrier is not empty.

```
lemma (in monoid0) group0_1_L3A: G≠0
proof -
  have TheNeutralElement(G,f) ∈ G using group0_1_L3
    by simp
  thus thesis by auto
qed
```

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

```
lemma (in monoid0) group0_1_L3B: range(f) = G
proof
  from monoidAsssum have T1:f : G×G→G
     using IsAmonoid_def IsAssociative_def by simp
  then show range(f) ⊆ G
    using func1_1_L5B by simp
  show G ⊆ range(f)
  proof
    fix g assume A1:g∈G
    let e = TheNeutralElement(G,f)
    from A1 have <e,g> ∈ G×G g = f<e,g>
      using group0_1_L3 by auto
    with T1 show g ∈ range(f)
      using func1_1_L5A by blast
  qed
qed
```

In a monoid a neutral element is the neutral element.

```
lemma (in monoid0) group0_1_L4:
  assumes A1: e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g)
  shows e = TheNeutralElement(G,f)
proof -
  let n =  THE b. b∈ G ∧ (∀  g∈G. b⊕g = g ∧ g⊕b = g)
  have ∃!b. b∈ G ∧ (∀  g∈G. b⊕g = g ∧ g⊕b = g)
    using group0_1_L2 by simp
  moreover from A1 have
    e ∈ G ∧ (∀g∈G. e ⊕ g = g ∧ g ⊕ e = g) .
  ultimately have (n) = e by (rule the_equality2)
  then show thesis using TheNeutralElement_def by simp
qed
```

The next lemma shows that if the if we restrict the monoid operation to a subset of G that contains the neutral element, then the neutral element of the monoid operation is also neutral with the restricted operation. This is proven separately because it is used more than once.

```
lemma (in monoid0) group0_1_L5:
  assumes A1: ∀x∈H.∀y∈H. x⊕y ∈ H
  and A2: H⊆G
  and A3: e = TheNeutralElement(G,f)
  and A4: g = restrict(f,H×H)
  and A5: e∈H
  and A6: h∈H
  shows g<e,h> = h ∧ g<h,e> = h
proof -
  from A4 A6 A5 have
    g<e,h> = e⊕h ∧ g<h,e> = h⊕e
    using restrict_if by simp
  with A3 A4 A6 A2 show
    g<e,h> = h ∧ g<h,e> = h
    using  group0_1_L3 by auto
qed
```

The next theorem shows that if the monoid operation is closed on a subset of G then this set is a (sub)monoid. (although we do not define this notion). This will be useful when we study subgroups.

```
theorem (in monoid0) group0_1_T1:
  assumes A1: H {is closed under} f
  and A2: H⊆G
  and A3: TheNeutralElement(G,f) ∈ H
  shows  IsAmonoid(H,restrict(f,H×H))
proof -
  let g = restrict(f,H×H)
  let e = TheNeutralElement(G,f)
  from monoidAsssum have f ∈ G×G→G
    using IsAmonoid_def IsAssociative_def by simp
  moreover from A2 have H×H ⊆ G×G by auto
```

```
  moreover from A1 have ∀p ∈ H×H. f(p) ∈ H
    using IsOpClosed_def by auto
  ultimately have g ∈ H×H→H
    using func1_2_L4 by simp
  moreover have ∀x∈H.∀y∈H.∀z∈H.
    g⟨g<x,y>,z⟩ = g⟨x,g<y,z>⟩
  proof -
    from A1 have ∀x∈H.∀y∈H.∀z∈H.
      g⟨g<x,y>,z⟩ = x⊕y⊕z
      using IsOpClosed_def restrict_if by simp
    moreover have ∀x∈H.∀y∈H.∀z∈H. x⊕y⊕z = x⊕(y⊕z)
    proof -
      from monoidAsssum have
        ∀x∈G.∀y∈G.∀z∈G. x⊕y⊕z = x⊕(y⊕z)
        using IsAmonoid_def IsAssociative_def
        by simp
      with A2 show thesis by auto
    qed
    moreover from A1 have
      ∀x∈H.∀y∈H.∀z∈H. x⊕(y⊕z) = g⟨ x,g<y,z>⟩
      using IsOpClosed_def restrict_if by simp
    ultimately show thesis by simp
  qed
  moreover have
    ∃n∈H. (∀h∈H. g<n,h> = h ∧ g<h,n> = h)
  proof -
    from A1 have ∀x∈H.∀y∈H. x⊕y ∈ H
      using IsOpClosed_def by simp
    with A2 A3 have
      ∀ h∈H. g<e,h> = h ∧ g<h,e> = h
      using group0_1_L5 by blast
    with A3 show thesis by auto
  qed
  ultimately show thesis using IsAmonoid_def IsAssociative_def
    by simp
qed
```

Under the assumptions of group0_1_T1 the neutral element of a submonoid is the same as that of the monoid.

```
lemma group0_1_L6:
  assumes A1: IsAmonoid(G,f)
  and A2: H {is closed under} f
  and A3: H⊆G
  and A4: TheNeutralElement(G,f) ∈ H
  shows TheNeutralElement(H,restrict(f,H×H)) = TheNeutralElement(G,f)
proof -
  def D1: e ≡ TheNeutralElement(G,f)
  def D2: g ≡ restrict(f,H×H)
  with A1 A2 A3 A4 have monoid0(H,g)
```

```
    using monoid0_def monoid0.group0_1_T1
    by simp
  moreover have
    e ∈ H ∧ (∀h∈H. g<e,h> = h ∧ g<h,e> = h)
  proof -
    from A1 A2 have monoid0(G,f) ∀x∈H.∀y∈H. f<x,y> ∈ H
      using monoid0_def IsOpClosed_def by auto
    with A3 D1 D2 A4 show thesis
      using monoid0.group0_1_L5 by blast
  qed
  ultimately have e =  TheNeutralElement(H,g)
    using monoid0.group0_1_L4 by auto
  with D1 D2 show thesis by simp
qed
```

13.2 Basic definitions and results for 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. We also define the group inverse as a relation on the group carrier. Later we will show that this relation is a function. The `GroupInv` below is really the right inverse, understood as a function, that is a subset of $G \times G$.

```
constdefs
  IsAgroup(G,f) ≡
  (IsAmonoid(G,f) ∧ (∀g∈G. ∃b∈G. f<g,b> = TheNeutralElement(G,f)))

  GroupInv(G,f) ≡ {<x,y> ∈ G×G. f<x,y> = TheNeutralElement(G,f)}
```

We will use the miltiplicative notation for groups.

```
locale group0 =
  fixes G and f
  assumes groupAssum: IsAgroup(G,f)

  fixes neut (1)
  defines neut_def[simp]: 1 ≡ TheNeutralElement(G,f)

  fixes groper (infixl · 70)
  defines groper_def [simp]: a · b ≡ f<a,b>

  fixes inv (_⁻¹  [90] 91)
  defines inv_def[simp]: x⁻¹ ≡ GroupInv(G,f)(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: monoid0(G,f)
  using groupAssum IsAgroup_def monoid0_def by simp
```

In some strange cases Isabelle has difficulties with applying the definition of a group. The next lemma defines a rule to be applied in such cases.

```
lemma definition_of_group: assumes IsAmonoid(G,f)
  and ∀g∈G. ∃b∈G. f⟨g,b⟩ = TheNeutralElement(G,f)
  shows IsAgroup(G,f)
  using prems IsAgroup_def by simp
```

Technical lemma that allows to use 1 as the neutral element of the group without referencing a list of lemmas and definitions.

```
lemma (in group0) group0_2_L2:
  shows 1∈G ∧ (∀g∈G.(1·g = g ∧ g·1 = g))
  using group0_2_L1 monoid0.group0_1_L3 by simp
```

The group is closed under the group operation. Used all the time, useful to have handy.

```
lemma (in group0) group_op_closed: assumes a∈G  b∈G
  shows a·b ∈ G using prems group0_2_L1 monoid0.group0_1_L1
  by simp
```

The group operation is associative. This is another technical lemma that allows to shorten the list of referenced lemmas in some proofs.

```
lemma (in group0) group_oper_assoc:
  assumes a∈G  b∈G  c∈G shows a·(b·c) = a·b·c
  using groupAssum prems IsAgroup_def IsAmonoid_def
    IsAssociative_def group_op_closed by simp
```

The group operation maps $G \times G$ into G. It is conveniet to have this fact easily accessible in the group0 context.

```
lemma (in group0) group_oper_assocA: shows f : G×G→G
  using groupAssum IsAgroup_def IsAmonoid_def IsAssociative_def
  by simp
```

The definition of group requires the existence of the right inverse. We show that this is also the left inverse.

```
theorem (in group0) group0_2_T1:
  assumes A1: g∈G and A2: b∈G and A3: g·b = 1
  shows b·g = 1
proof -
  from A2 groupAssum obtain c where I: c ∈ G ∧ b·c = 1
    using IsAgroup_def by auto
  then have T1: c∈G by simp
  have T2: 1∈G using group0_2_L2 by simp
  from A1 A2 T2 I have b·g =  b·(g·(b·c))
    using group_op_closed group0_2_L2 group_oper_assoc
    by simp
  also from  A1 A2 T1 have b·(g·(b·c)) = b·(g·b·c)
    using group_oper_assoc by simp
```

```
  also from A3 A2 I have b·(g·b·c)= 1 using group0_2_L2 by simp
  finally show b·g = 1 by simp
qed
```

For every element of a group there is only one inverse.

```
lemma (in group0) group0_2_L4:
  assumes A1:x∈G shows ∃!y. y∈G ∧ x·y = 1
proof
  from A1 groupAssum show ∃y. y∈G ∧  x·y = 1
    using IsAgroup_def by auto
  fix y n
  assume A2:y∈G ∧  x·y = 1 and A3:n∈G ∧ x·n = 1 show y=n
  proof -
    from A1 A2 have T1:y·x = 1
      using group0_2_T1 by simp
    from A2 A3 have y = y·(x·n)
      using group0_2_L2 by simp
    also from A1 A2 A3 have ... = (y·x)·n
      using group_oper_assoc by blast
    also from T1 A3 have ... = n
      using group0_2_L2 by simp
    finally show y=n by simp
  qed
qed
```

The group inverse is a function that maps G into G.

```
theorem group0_2_T2:
  assumes A1: IsAgroup(G,f) shows GroupInv(G,f) : G→G
proof -
  have GroupInv(G,f) ⊆ G×G using GroupInv_def by auto
  moreover from A1 have
    ∀x∈G. ∃!y. y∈ G ∧ <x,y> ∈ GroupInv(G,f)
    using group0_def group0.group0_2_L4 GroupInv_def by simp
  ultimately show thesis using func1_1_L11 by simp
qed
```

We can think about the group inverse (the function) as the inverse image of the neutral element.

```
theorem (in group0) group0_2_T3: shows f-{1} = GroupInv(G,f)
proof -
  from groupAssum have f : G×G → G
    using IsAgroup_def IsAmonoid_def IsAssociative_def
    by simp
  then show f-{1} = GroupInv(G,f)
    using func1_1_L14 GroupInv_def by auto
qed
```

The inverse is in the group.

```
lemma (in group0) inverse_in_group: assumes A1: x∈G shows x⁻¹∈G
```

```
proof -
  from groupAssum have GroupInv(G,f) : G→G using group0_2_T2 by simp
  with A1 show thesis using apply_type by simp
qed
```

The notation for the inverse means what it is supposed to mean.

```
lemma (in group0) group0_2_L6:
  assumes A1: x∈G shows x·x⁻¹ = 1 ∧ x⁻¹·x = 1
proof
  from groupAssum have GroupInv(G,f) : G→G
    using group0_2_T2 by simp
  with A1 have <x,x⁻¹> ∈  GroupInv(G,f)
    using apply_Pair by simp
  then show x·x⁻¹ = 1 using GroupInv_def by simp
  with A1 show x⁻¹·x = 1 using inverse_in_group group0_2_T1 by blast

qed
```

The next two lemmas state that unless we multiply by the neutral element, the result is always different than any of the operands.

```
lemma (in group0) group0_2_L7:
  assumes A1: a∈G and A2: b∈G and A3: a·b = a
  shows b=1
proof -
  from A3 have a⁻¹ · (a·b) = a⁻¹·a by simp
  with A1 A2 show thesis using
    inverse_in_group group_oper_assoc group0_2_L6 group0_2_L2
    by simp
qed

lemma (in group0) group0_2_L8:
  assumes A1: a∈G and A2: b∈G and A3:a·b = b
  shows a=1
proof -
  from A3 have (a·b)·b⁻¹  = b·b⁻¹ by simp
  with A1 A2 have a·(b·b⁻¹)  = b·b⁻¹ using
    inverse_in_group group_oper_assoc by simp
  with A1 A2 show thesis
    using group0_2_L6 group0_2_L2 by simp
qed
```

The inverse of the neutral element is the neutral element.

```
lemma (in group0) group_inv_of_one: shows 1⁻¹=1
  using group0_2_L2 inverse_in_group group0_2_L6 group0_2_L7 by blast
```

if $a^{-1} = 1$, then $a = 1$.

```
lemma (in group0) group0_2_L8A:
  assumes A1: a∈G and A2: a⁻¹ = 1
```

```
  shows a = 1
proof -
  from A1 have a·a⁻¹ = 1 using group0_2_L6 by simp
  with A1 A2 show a = 1 using group0_2_L2 by simp
qed
```

If a is not a unit, then its inverse is not either.

```
lemma (in group0) group0_2_L8B:
  assumes a∈G and a ≠ 1
  shows a⁻¹ ≠ 1 using prems group0_2_L8A by auto
```

If a^{-1} is not a unit, then a is not either.

```
lemma (in group0) group0_2_L8C:
  assumes a∈G and a⁻¹ ≠ 1
  shows a≠1
  using prems group0_2_L8A group_inv_of_one by auto
```

If a product of two elements of a group is equal to the neutral element then they are inverses of each other.

```
lemma (in group0) group0_2_L9:
  assumes A1: a∈G and A2: b∈G and A3: a·b = 1
  shows a = b⁻¹  b = a⁻¹
proof -
  from A3 have a·b·b⁻¹ = 1·b⁻¹ by simp
  with A1 A2 have a·(b·b⁻¹) = 1·b⁻¹ using
    inverse_in_group group_oper_assoc by simp
  with A1 A2 show a = b⁻¹ using
    group0_2_L6 inverse_in_group group0_2_L2 by simp
  from A3 have a⁻¹·(a·b) = a⁻¹·1 by simp
  with A1 A2 show b = a⁻¹ using
    inverse_in_group group_oper_assoc group0_2_L6 group0_2_L2
    by simp
qed
```

It happens quite often that we know what is (have a meta-function for) the right inverse in a group. The next lemma shows that the value of the group inverse (function) is equal to the right inverse (meta-function).

```
lemma (in group0) group0_2_L9A:
  assumes A1: ∀g∈G. b(g) ∈ G ∧ g·b(g) = 1
  shows ∀g∈G. b(g) = g⁻¹
proof
  fix g assume A2: g∈G
  moreover from A2 A1 have b(g) ∈ G by simp
  moreover from A1 A2 have g·b(g) = 1 by simp
  ultimately show b(g) = g⁻¹ by (rule group0_2_L9)
qed
```

What is the inverse of a product?

```
lemma (in group0) group_inv_of_two:
  assumes A1: a∈G and A2: b∈G
  shows  b⁻¹·a⁻¹ = (a·b)⁻¹
proof -
  from A1 A2 have
    T1: b⁻¹∈G and T2: a⁻¹∈G and T3: a·b∈G and T4: b⁻¹·a⁻¹ ∈ G
    using inverse_in_group group_op_closed
    by auto
  from A1 A2 T4 have a·b·(b⁻¹·a⁻¹) = a·(b·(b⁻¹·a⁻¹))
    using group_oper_assoc by simp
  moreover from A2 T1 T2 have b·(b⁻¹·a⁻¹) = b·b⁻¹·a⁻¹
    using group_oper_assoc by simp
  moreover from A2 T2 have b·b⁻¹·a⁻¹ = a⁻¹
     using group0_2_L6 group0_2_L2 by simp
  ultimately have a·b·(b⁻¹·a⁻¹) = a·a⁻¹
    by simp
  with A1 have a·b·(b⁻¹·a⁻¹) = 1
    using group0_2_L6 by simp
  with T3 T4 show b⁻¹·a⁻¹ = (a·b)⁻¹
    using group0_2_L9 by simp
qed
```

What is the inverse of a product of three elements?

```
lemma (in group0) group_inv_of_three:
  assumes A1: a∈G  b∈G  c∈G
  shows
  (a·b·c)⁻¹ = c⁻¹·(a·b)⁻¹
  (a·b·c)⁻¹ = c⁻¹·(b⁻¹·a⁻¹)
  (a·b·c)⁻¹ = c⁻¹·b⁻¹·a⁻¹
proof -
  from A1 have T:
    a·b ∈ G a⁻¹ ∈ G  b⁻¹ ∈ G   c⁻¹ ∈ G
    using group_op_closed inverse_in_group by auto
  with A1 show
    (a·b·c)⁻¹ = c⁻¹·(a·b)⁻¹ and (a·b·c)⁻¹ = c⁻¹·(b⁻¹·a⁻¹)
     using group_inv_of_two by auto
   with T show (a·b·c)⁻¹ = c⁻¹·b⁻¹·a⁻¹ using group_oper_assoc
     by simp
qed
```

The inverse of the inverse is the element.

```
lemma (in group0) group_inv_of_inv:
  assumes a∈G shows a = (a⁻¹)⁻¹
  using prems inverse_in_group group0_2_L6 group0_2_L9
  by simp
```

If $a^{-1} \cdot b = 1$, then $a = b$.

```
lemma (in group0) group0_2_L11:
  assumes A1: a∈G  b∈G and A2: a⁻¹·b = 1
```

```
  shows a=b
proof -
  from A1 A2 have a⁻¹ ∈ G  b∈G  a⁻¹·b = 1
    using inverse_in_group by auto
  then have b = (a⁻¹)⁻¹ by (rule group0_2_L9)
  with A1 show a=b using group_inv_of_inv by simp
qed
```

If $a \cdot b^{-1} = 1$, then $a = b$.

```
lemma (in group0) group0_2_L11A:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ = 1
  shows a=b
proof -
  from A1 A2 have a ∈ G  b⁻¹∈G  a·b⁻¹ = 1
    using inverse_in_group by auto
  then have a = (b⁻¹)⁻¹ by (rule group0_2_L9)
  with A1 show a=b using group_inv_of_inv by simp
qed
```

If if the inverse of b is different than a, then the inverse of a is different than b.

```
lemma (in group0) group0_2_L11B:
  assumes A1: a∈G and A2: b⁻¹ ≠ a
  shows a⁻¹ ≠ b
proof -
  { assume a⁻¹ = b
    then have (a⁻¹)⁻¹ = b⁻¹ by simp
    with A1 A2 have False using group_inv_of_inv
      by simp
  } then show a⁻¹ ≠ b by auto
qed
```

What is the inverse of ab^{-1} ?

```
lemma (in group0) group0_2_L12:
  assumes A1: a∈G  b∈G
  shows
  (a·b⁻¹)⁻¹ = b·a⁻¹
  (a⁻¹·b)⁻¹ = b⁻¹·a
proof -
  from A1 have
    (a·b⁻¹)⁻¹ = (b⁻¹)⁻¹· a⁻¹  (a⁻¹·b)⁻¹ = b⁻¹·(a⁻¹)⁻¹
    using inverse_in_group group_inv_of_two by auto
  with A1 show  (a·b⁻¹)⁻¹ = b·a⁻¹  (a⁻¹·b)⁻¹ = b⁻¹·a
    using group_inv_of_inv by auto
qed
```

A couple useful rearrangements with three elements: we can insert a $b \cdot b^{-1}$ between two group elements (another version) and one about a product of an element and inverse of a product, and two others.

```
lemma (in group0) group0_2_L14A:
  assumes A1: a∈G  b∈G  c∈G
  shows
  a·c⁻¹= (a·b⁻¹)·(b·c⁻¹)
  a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
  a·(b·c)⁻¹ = a·c⁻¹·b⁻¹
  a·(b·c⁻¹) = a·b·c⁻¹
  (a·b⁻¹·c⁻¹)⁻¹ = c·b·a⁻¹
  a·b·c⁻¹·(c·b⁻¹) = a
  a·(b·c)·c⁻¹ = a·b
proof -
  from A1 have T:
    a⁻¹ ∈ G  b⁻¹∈G  c⁻¹∈G
    a⁻¹·b ∈ G  a·b⁻¹ ∈ G  a·b ∈ G
    c·b⁻¹ ∈ G  b·c ∈ G
    using inverse_in_group group_op_closed
    by auto
   from A1 T have
     a·c⁻¹ =  a·(b⁻¹·b)·c⁻¹
     a⁻¹·c =  a⁻¹·(b·b⁻¹)·c
    using group0_2_L2 group0_2_L6 by auto
   with A1 T show
     a·c⁻¹= (a·b⁻¹)·(b·c⁻¹)
     a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
     using group_oper_assoc by auto
  from A1 have a·(b·c)⁻¹ = a·(c⁻¹·b⁻¹)
    using group_inv_of_two by simp
  with A1 T show a·(b·c)⁻¹ =a·c⁻¹·b⁻¹
    using group_oper_assoc by simp
  from A1 T show a·(b·c⁻¹) = a·b·c⁻¹
    using group_oper_assoc by simp
  from A1 T show  (a·b⁻¹·c⁻¹)⁻¹ = c·b·a⁻¹
    using group_inv_of_three  group_inv_of_inv
    by simp
  from T have a·b·c⁻¹·(c·b⁻¹) = a·b·(c⁻¹·(c·b⁻¹))
    using group_oper_assoc by simp
  also from A1 T have ... =  a·b·b⁻¹
    using group_oper_assoc group0_2_L6 group0_2_L2
    by simp
  also from A1 T have ... = a·(b·b⁻¹)
    using group_oper_assoc by simp
  also from A1 have ... = a
    using group0_2_L6 group0_2_L2 by simp
  finally show a·b·c⁻¹·(c·b⁻¹) = a by simp
  from A1 T have a·(b·c)·c⁻¹ =  a·(b·(c·c⁻¹))
    using group_oper_assoc by simp
  also from A1 T have ... = a·b
    using  group0_2_L6 group0_2_L2 by simp
  finally show a·(b·c)·c⁻¹ = a·b
```

```
    by simp
qed
```

Another lemma about rearranging a product.

```
lemma (in group0) group0_2_L15:
  assumes A1: a∈G b∈G c∈G d∈G
  shows (a·b)·(c·d)⁻¹ = a·(b·d⁻¹)·a⁻¹·(a·c⁻¹)
proof -
  from A1 have T1:
    d⁻¹∈G  c⁻¹∈G a·b∈G a·(b·d⁻¹)∈G
    using inverse_in_group group_op_closed
    by auto
  with A1 have (a·b)·(c·d)⁻¹ = (a·b)·(d⁻¹·c⁻¹)
    using group_inv_of_two by simp
  also from A1 T1 have ... = a·(b·d⁻¹)·c⁻¹
    using group_oper_assoc by simp
  also from A1 T1 have ... = a·(b·d⁻¹)·a⁻¹·(a·c⁻¹)
    using group0_2_L14A by blast
  finally show thesis by simp
qed
```

We can cancel an element with its inverse that is written next to it.

```
lemma (in group0) group0_2_L16:
  assumes A1: a∈G  b∈G
  shows
  a·b⁻¹·b = a
  a·b·b⁻¹ = a
  a⁻¹·(a·b) = b
  a·(a⁻¹·b) = b
proof -
  from A1 have
    a·b⁻¹·b = a·(b⁻¹·b)   a·b·b⁻¹ = a·(b·b⁻¹)
    a⁻¹·(a·b) = a⁻¹·a·b   a·(a⁻¹·b) = a·a⁻¹·b
    using inverse_in_group group_oper_assoc by auto
  with A1 show
    a·b⁻¹·b = a
    a·b·b⁻¹ = a
    a⁻¹·(a·b) = b
    a·(a⁻¹·b) = b
    using group0_2_L6 group0_2_L2 by auto
qed
```

Another lemma about cancelling with two group elements.

```
lemma (in group0) group0_2_L16A:
  assumes A1: a∈G  b∈G
  shows a·(b·a)⁻¹ = b⁻¹
proof -
  from A1 have (b·a)⁻¹ = a⁻¹·b⁻¹  b⁻¹ ∈ G
    using group_inv_of_two inverse_in_group by auto
```

```
  with A1 show a·(b·a)⁻¹ = b⁻¹ using group0_2_L16
    by simp
qed
```

A hard to clasify fact: 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 A1: H⊆G
  and A2: H {is closed under} f
  shows (H ∪ {1}) {is closed under} f
proof -
  { fix a b assume A3: a ∈ H ∪ {1}  b ∈ H ∪ {1}
    have a·b ∈ H ∪ {1}
    proof (cases a∈H)
      assume A4: a∈H show a·b ∈ H ∪ {1}
      proof (cases b∈H)
        assume b∈H
        with A2 A4 show a·b ∈ H ∪ {1} using IsOpClosed_def
          by simp
      next assume b∉H
        with A1 A3 A4 show a·b ∈ H ∪ {1}
          using group0_2_L2 by auto
      qed
    next assume a∉H
      with A1 A3 show a·b ∈ H ∪ {1}
        using group0_2_L2 by auto
    qed
  } then show (H ∪ {1}) {is closed under} f
    using IsOpClosed_def by auto
qed
```

We can put an element on the other side of an equation.

```
lemma (in group0) group0_2_L18:
  assumes A1: a∈G  b∈G  c∈G
  and A2: c = a·b
  shows c·b⁻¹ = a  a⁻¹·c = b
proof-
  from A2 A1 have c·b⁻¹ =  a·(b·b⁻¹)  a⁻¹·c = (a⁻¹·a)·b
    using inverse_in_group group_oper_assoc by auto
  moreover from A1 have a·(b·b⁻¹) = a  (a⁻¹·a)·b = b
    using group0_2_L6 group0_2_L2 by auto
  ultimately show c·b⁻¹ = a  a⁻¹·c = b
    by auto
qed
```

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
  c·a ≠ c·b
proof -
  { assume a·c = b·c ∨ c·a =c·b
    then have a·c·c⁻¹ = b·c·c⁻¹ ∨ c⁻¹·(c·a) = c⁻¹·(c·b)
      by auto
    with A1 A2 have False using group0_2_L16 by simp
  } then show a·c ≠ b·c and c·a ≠ c·b by auto
qed
```

13.3 Subgroups

There are two common ways to define subgroups. One requires that the group operations are 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. We do not require H to be a subset of G as this can be inferred from our definition. The rest of this section is devoted to proving the equivalence of these two definitions of the notion of a subgroup.

```
constdefs
  IsAsubgroup(H,f) ≡ IsAgroup(H, restrict(f,H×H))
```

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 neutral element of the subgroup is in the subgroup and it is both right and left neutral there. The notation is very ugly because we don't want to introduce a separate notation for the subgroup operation.

```
lemma group0_3_L1:
  assumes A1:IsAsubgroup(H,f)
  and A2: n = TheNeutralElement(H,restrict(f,H×H))
  shows n ∈ H
  ∀h∈H. restrict(f,H×H)<n,h > = h
  ∀h∈H. restrict(f,H×H)<h,n > = h
proof -
  let b = restrict(f,H×H)
  let e = TheNeutralElement(H,restrict(f,H×H))
  from A1 have group0(H,b)
    using IsAsubgroup_def group0_def by simp
  then have T1:
    e ∈ H ∧ (∀h∈H. (b<e,h > = h ∧ b<h,e > = h))
    by (rule group0.group0_2_L2)
  with A2 show n ∈ H by simp
```

```
  from T1 A2 show ∀h∈H. b<n,h > = h ∀h∈H. b<h,n> = h
    by auto
qed
```

Subgroup is contained in the group.

```
lemma (in group0) group0_3_L2:
  assumes A1:IsAsubgroup(H,f)
  shows H⊆G
proof
  fix h assume A2:h∈H
  let b = restrict(f,H×H)
  let n = TheNeutralElement(H,restrict(f,H×H))
   from A1 have b ∈ H×H→H
    using IsAsubgroup_def IsAgroup_def
      IsAmonoid_def IsAssociative_def by simp
  moreover from A2 A1 have <n,h> ∈ H×H
    using group0_3_L1 by simp
  moreover from A1 A2 have h = b<n,h >
    using group0_3_L1 by simp
  ultimately have ⟨<n,h>,h⟩ ∈ b
    using func1_1_L5A by blast
  then have ⟨<n,h>,h⟩ ∈ f using restrict_subset by auto
  moreover from groupAssum have f:G×G→G
    using IsAgroup_def IsAmonoid_def IsAssociative_def
    by simp
  ultimately show h∈G using func1_1_L5
    by blast
qed
```

The group neutral element (denoted 1 in the group0 context) is a neutral element for the subgroup with respect to the froup action.

```
lemma (in group0) group0_3_L3:
  assumes A1:IsAsubgroup(H,f)
  shows ∀h∈H. 1·h = h ∧ h·1 = h
proof
  fix h assume h∈H
  with groupAssum A1 show 1·h = h ∧ h·1 = h
    using group0_3_L2 group0_2_L2 by auto
qed
```

The neutral element of a subgroup is the same as that of the group.

```
lemma (in group0) group0_3_L4: assumes A1:IsAsubgroup(H,f)
  shows TheNeutralElement(H,restrict(f,H×H)) = 1
proof -
  let n =  TheNeutralElement(H,restrict(f,H×H))
  from A1 have T1:n ∈ H using group0_3_L1 by simp
  with groupAssum A1 have n∈G using  group0_3_L2 by auto
  with A1 T1 show thesis using
     group0_3_L1 restrict_if group0_2_L7 by simp
```

```
qed
```

The neutral element of the group (denoted 1 in the group0 context) belongs to every subgroup.

```
lemma (in group0) group0_3_L5: assumes A1: IsAsubgroup(H,f)
  shows 1∈H
proof -
  from A1 show 1∈H using group0_3_L1 group0_3_L4
    by fast
qed
```

Subgroups are closed with respect to the group operation.

```
lemma (in group0) group0_3_L6: assumes A1:IsAsubgroup(H,f)
  and A2:a∈H b∈H
  shows a·b ∈ H
proof -
  let b = restrict(f,H×H)
  from A1 have monoid0(H,b) using
    IsAsubgroup_def IsAgroup_def monoid0_def by simp
  with A2 have b (<a,b>) ∈ H using monoid0.group0_1_L1
    by blast
 with A2 show a·b ∈ H using restrict_if by simp
qed
```

A preliminary lemma that we need to show that taking the inverse in the subgroup is the same as taking the inverse in the group.

```
lemma group0_3_L7A:
  assumes A1:IsAgroup(G,f)
  and A2:IsAsubgroup(H,f) and A3:g=restrict(f,H×H)
  shows GroupInv(G,f) ∩ H×H = GroupInv(H,g)
proof -
  def D1: e ≡ TheNeutralElement(G,f)
  def D2: e1 ≡ TheNeutralElement(H,g)
  from A1 have T1:group0(G,f) using group0_def by simp
  from A2 A3 have T2:group0(H,g)
    using IsAsubgroup_def group0_def by simp
  from T1 A2 A3 D1 D2 have e1 = e
    using group0.group0_3_L4 by simp
  with T1 D1 have GroupInv(G,f) = f-{e1}
    using group0.group0_2_T3 by simp
  moreover have g-{e1} = f-{e1} ∩ H×H
  proof -
    from A1 have f ∈ G×G→G
      using IsAgroup_def IsAmonoid_def IsAssociative_def
      by simp
    moreover from T1 A2 have H×H ⊆ G×G
      using group0.group0_3_L2 by auto
    ultimately show g-{e1} = f-{e1} ∩ H×H
```

```
    using A3 func1_2_L1 by simp
  qed
  moreover from T2 A3 D2 have GroupInv(H,g) = g-{e1}
    using group0.group0_2_T3 by simp
  ultimately show thesis by simp
qed
```

Using the lemma above we can show the actual statement: taking the inverse in the subgroup is the same as taking the inverse in the group.

```
theorem (in group0) group0_3_T1:
  assumes A1: IsAsubgroup(H,f)
  and A2:g=restrict(f,H×H)
  shows GroupInv(H,g) = restrict(GroupInv(G,f),H)
proof -
  from groupAssum have GroupInv(G,f) : G→G
    using group0_2_T2 by simp
  moreover from A1 A2 have GroupInv(H,g) : H→H
    using IsAsubgroup_def group0_2_T2 by simp
  moreover from A1 have H⊆G
    using group0_3_L2 by simp
  moreover from groupAssum A1 A2 have
    GroupInv(G,f) ∩ H×H = GroupInv(H,g)
    using group0_3_L7A by simp
  ultimately show thesis
    using func1_2_L3 by simp
qed
```

A sligtly weaker, but more convenient in applications, reformulation of the above theorem.

```
theorem (in group0) group0_3_T2:
  assumes IsAsubgroup(H,f)
  and g=restrict(f,H×H)
  shows ∀h∈H. GroupInv(H,g)(h) = h⁻¹
  using prems group0_3_T1 restrict_if by simp
```

Subgroups are closed with respect to taking the group inverse. Again, I was unable to apply `inverse_in_group` directly to the group H. This problem is worked around by repeating the (short) proof of `inverse_in_group` in the proof below.

```
theorem (in group0) group0_3_T3A:
  assumes A1:IsAsubgroup(H,f) and A2:h∈H
  shows h⁻¹∈ H
proof -
  def D1: g ≡ restrict(f,H×H)
  with A1 have  GroupInv(H,g) ∈ H→H
    using IsAsubgroup_def group0_2_T2 by simp
  with A2 have GroupInv(H,g)(h) ∈ H
    using apply_type by simp
```

```
  with A1 D1 A2 show h⁻¹∈ H using group0_3_T2 by simp
qed
```

The next theorem states that a nonempty subset of 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} f
  and A4: ∀x∈H. x⁻¹ ∈ H
  shows IsAsubgroup(H,f)
proof -
  let g = restrict(f,H×H)
  let n = TheNeutralElement(H,g)
  from A3 have T0:∀x∈H.∀y∈H. x·y ∈ H
    using IsOpClosed_def by simp
  from A1 obtain x where x∈H by auto
  with A4 T0 A2 have T1:1∈H
    using group0_2_L6 by blast
  with A3 A2 have T2:IsAmonoid(H,g)
    using group0_2_L1 monoid0.group0_1_T1
    by simp
  moreover have ∀h∈H.∃b∈H. g<h,b> = n
  proof
    fix h assume A5:h∈H
    with A4 A2 have h·h⁻¹ = 1
      using group0_2_L6 by auto
    moreover from groupAssum A3 A2 T1 have 1 = n
      using IsAgroup_def group0_1_L6 by auto
    moreover from A5 A4 have g<h,h⁻¹> = h·h⁻¹
      using restrict_if by simp
    ultimately have g<h,h⁻¹> = n by simp
    with A5 A4 show ∃b∈H. g<h,b> = n by auto
  qed
  ultimately show IsAsubgroup(H,f) using
    IsAsubgroup_def IsAgroup_def by simp
qed
```

Intersection of subgroups is a subgroup of each factor.

```
lemma group0_3_L7:
  assumes A1:IsAgroup(G,f)
  and A2:IsAsubgroup(H₁,f)
  and A3:IsAsubgroup(H₂,f)
  shows IsAsubgroup(H₁∩H₂,restrict(f,H₁×H₁))
proof -
  let e = TheNeutralElement(G,f)
  let g = restrict(f,H₁×H₁)
  from A1 have T1: group0(G,f)
```

```
    using group0_def by simp
  from A2 have group0(H₁,g)
    using IsAsubgroup_def group0_def by simp
  moreover have H₁∩H₂ ≠ 0
  proof -
    from A1 A2 A3 have e ∈ H₁∩H₂
      using group0_def group0.group0_3_L5 by simp
    thus thesis by auto
  qed
  moreover have T2:H₁∩H₂ ⊆ H₁ by auto
  moreover from T1 T2 A2 A3 have
    H₁∩H₂ {is closed under} g
    using group0.group0_3_L6 IsOpClosed_def
      func_ZF_4_L7 func_ZF_4_L5 by simp
  moreover from T1 A2 A3 have
    ∀x ∈ H₁∩H₂. GroupInv(H₁,g)(x) ∈ H₁∩H₂
    using group0.group0_3_T2 group0.group0_3_T3A
    by simp
  ultimately show thesis
    using group0.group0_3_T3 by simp
qed
```

13.4 Abelian groups

Here we will prove some facts specific to abelian groups.

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. For the parantheses simp does it very well.

```
lemma (in group0) group0_4_L2:
  assumes A1:f {is commutative on} G
  and A2:a∈G b∈G c∈G d∈G E∈G F∈G
  shows (a·b)·(c·d)·(E·F) = (a·(d·F))·(b·(c·E))
proof -
  from A2 have (a·b)·(c·d)·(E·F) = a·b·c·d·E·F
    using group_op_closed group_oper_assoc
    by simp
  also have  a·b·c·d·E·F = a·d·F·b·c·E
  proof -
    from A1 A2 have a·b·c·d·E·F = F·(a·b·c·d·E)
```

```
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have F·(a·b·c·d·E) = F·a·b·c·d·E
      using group_op_closed group_oper_assoc
      by simp
    also from A1 A2 have F·a·b·c·d·E = d·(F·a·b·c)·E
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have d·(F·a·b·c)·E = d·F·a·b·c·E
      using group_op_closed group_oper_assoc
      by simp
    also from A1 A2 have  d·F·a·b·c·E = a·(d·F)·b·c·E
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have a·(d·F)·b·c·E = a·d·F·b·c·E
      using group_op_closed group_oper_assoc
      by simp
    finally show thesis by simp
  qed
  also from A2 have a·d·F·b·c·E = (a·(d·F))·(b·(c·E))
    using group_op_closed group_oper_assoc
    by simp
  finally show thesis by simp
qed
```

Another useful rearrangement.

```
lemma (in group0) group0_4_L3:
  assumes A1:f {is commutative on} G
  and A2: a∈G  b∈G and A3: c∈G  d∈G  E∈G  F∈G
  shows a·b·((c·d)⁻¹·(E·F)⁻¹) = (a·(E·c)⁻¹)·(b·(F·d)⁻¹)
proof -
  from A3 have T1:
    c⁻¹∈G d⁻¹∈G E⁻¹∈G F⁻¹∈G (c·d)⁻¹∈G (E·F)⁻¹∈G
    using inverse_in_group group_op_closed
    by auto
  from A2 T1 have
    a·b·((c·d)⁻¹·(E·F)⁻¹) = a·b·(c·d)⁻¹·(E·F)⁻¹
    using group_op_closed group_oper_assoc
    by simp
  also from A2 A3 have
    a·b·(c·d)⁻¹·(E·F)⁻¹ = (a·b)·(d⁻¹·c⁻¹)·(F⁻¹·E⁻¹)
    using group_inv_of_two by simp
   also from A1 A2 T1 have
    (a·b)·(d⁻¹·c⁻¹)·(F⁻¹·E⁻¹) = (a·(c⁻¹·E⁻¹))·(b·(d⁻¹·F⁻¹))
    using group0_4_L2 by simp
  also from A2 A3 have
    (a·(c⁻¹·E⁻¹))·(b·(d⁻¹·F⁻¹)) = (a·(E·c)⁻¹)·(b·(F·d)⁻¹)
    using group_inv_of_two by simp
  finally show thesis by simp
```

qed

Some useful rearrangements for two elements of a group.

```
lemma (in group0) group0_4_L4:
  assumes A1:f {is commutative on} G
  and A2: a∈G b∈G
  shows
  b⁻¹·a⁻¹ = a⁻¹·b⁻¹
  (a·b)⁻¹ = a⁻¹·b⁻¹
  (a·b⁻¹)⁻¹ = a⁻¹·b
proof -
  from A2 have T1: b⁻¹∈G a⁻¹∈G using inverse_in_group by auto
  with A1 show b⁻¹·a⁻¹ = a⁻¹·b⁻¹ using IsCommutative_def by simp
  with A2 show (a·b)⁻¹ = a⁻¹·b⁻¹ using group_inv_of_two by simp
  from A2 T1 have (a·b⁻¹)⁻¹ = (b⁻¹)⁻¹·a⁻¹ using group_inv_of_two by simp
  with A1 A2 T1 show (a·b⁻¹)⁻¹ = a⁻¹·b
    using group_inv_of_inv IsCommutative_def by simp
qed
```

Another bunch of useful rearrangements with three elements.

```
lemma (in group0) group0_4_L4A:
  assumes A1:f {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·c = c·a·b
  a⁻¹·(b⁻¹·c⁻¹)⁻¹ = (a·(b·c)⁻¹)⁻¹
  a·(b·c)⁻¹ = a·b⁻¹·c⁻¹
  a·(b·c⁻¹)⁻¹ = a·b⁻¹·c
  a·b⁻¹·c⁻¹ = a·c⁻¹·b⁻¹
proof -
  from A1 A2 have a·b·c = c·(a·b)
    using IsCommutative_def group_op_closed
    by simp
  with A2 show a·b·c = c·a·b using
     group_op_closed group_oper_assoc
    by simp
  from A2 have T:
    b⁻¹∈G  c⁻¹∈G  b⁻¹·c⁻¹ ∈ G  a·b ∈ G
    using inverse_in_group group_op_closed
    by auto
  with A1 A2 show a⁻¹·(b⁻¹·c⁻¹)⁻¹ = (a·(b·c)⁻¹)⁻¹
    using group_inv_of_two IsCommutative_def
    by simp
  from A1 A2 T have a·(b·c)⁻¹ = a·(b⁻¹·c⁻¹)
    using group_inv_of_two IsCommutative_def by simp
  with A2 T show a·(b·c)⁻¹ = a·b⁻¹·c⁻¹
    using group_oper_assoc by simp
  from A1 A2 T have a·(b·c⁻¹)⁻¹ = a·(b⁻¹·(c⁻¹)⁻¹)
    using group_inv_of_two IsCommutative_def by simp
```

```
  with A2 T show a·(b·c⁻¹)⁻¹ = a·b⁻¹·c
    using group_oper_assoc group_inv_of_inv by simp
  from A1 A2 T have a·b⁻¹·c⁻¹ = a·(c⁻¹·b⁻¹)
    using group_oper_assoc IsCommutative_def by simp
  with A2 T show a·b⁻¹·c⁻¹ = a·c⁻¹·b⁻¹
    using group_oper_assoc by simp
qed
```

Another useful rearrangement.

```
lemma (in group0) group0_4_L4B:
  assumes f {is commutative on} G
  and a∈G  b∈G  c∈G
  shows a·b⁻¹·(b·c⁻¹) = a·c⁻¹
  using prems inverse_in_group group_op_closed
    group0_4_L4 group_oper_assoc group0_2_L16 by simp
```

A couple of permutations of order for three alements.

```
lemma (in group0) group0_4_L4C:
  assumes A1: f {is commutative on} G
  and A2: a∈G b∈G c∈G
  shows
  a·b·c = c·a·b
  a·b·c = a·(c·b)
  a·b·c = c·(a·b)
  a·b·c = c·b·a
proof -
  from A1 A2 show I: a·b·c = c·a·b
    using group0_4_L4A by simp
  also from A1 A2 have c·a·b = a·c·b
    using IsCommutative_def by simp
  also from A2 have a·c·b = a·(c·b)
    using group_oper_assoc by simp
  finally show a·b·c = a·(c·b) by simp
  from A2 I show a·b·c = c·(a·b)
    using group_oper_assoc by simp
  also from A1 A2 have c·(a·b) = c·(b·a)
    using IsCommutative_def by simp
  also from A2 have c·(b·a) = c·b·a
    using group_oper_assoc by simp
  finally show a·b·c = c·b·a by simp
qed
```

Some rearangement with three elements and inverse.

```
lemma (in group0) group0_4_L4D:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a⁻¹·b⁻¹·c = c·a⁻¹·b⁻¹
  b⁻¹·a⁻¹·c = c·a⁻¹·b⁻¹
```

```
  (a⁻¹·b·c)⁻¹ = a·b⁻¹·c⁻¹
proof -
  from A2 have T:
    a⁻¹ ∈ G  b⁻¹ ∈ G  c⁻¹∈G
    using inverse_in_group by auto
  with A1 A2 show
    a⁻¹·b⁻¹·c = c·a⁻¹·b⁻¹
    b⁻¹·a⁻¹·c = c·a⁻¹·b⁻¹
    using  group0_4_L4A by auto
  from A1 A2 T show (a⁻¹·b·c)⁻¹ = a·b⁻¹·c⁻¹
    using group_inv_of_three group_inv_of_inv group0_4_L4C
    by simp
qed
```

Another rearrangement lemma with three elements and equation.

```
lemma (in group0) group0_4_L5: assumes A1:f {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  and A3: c = a·b⁻¹
  shows a = b·c
proof -
  from A2 A3 have c·(b⁻¹)⁻¹ = a
    using inverse_in_group group0_2_L18
    by simp
  with A1 A2 show thesis using
     group_inv_of_inv IsCommutative_def by simp
qed
```

In abelian groups we can cancel an element with its inverse even if separated by another element.

```
lemma (in group0) group0_4_L6A: assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G
  shows
  a·b·a⁻¹ = b
  a⁻¹·b·a = b
  a⁻¹·(b·a) = b
  a·(b·a⁻¹) = b
proof -
  from A1 A2 have
    a·b·a⁻¹ = a⁻¹·a·b
    using inverse_in_group group0_4_L4A by blast
  also from A2 have ... = b
    using group0_2_L6 group0_2_L2 by simp
  finally show a·b·a⁻¹ = b by simp
  from A1 A2 have
    a⁻¹·b·a = a·a⁻¹·b
    using inverse_in_group group0_4_L4A by blast
  also from A2 have ... = b
    using group0_2_L6 group0_2_L2 by simp
  finally show a⁻¹·b·a = b by simp
```

```
  moreover from A2 have a⁻¹·b·a = a⁻¹·(b·a)
    using inverse_in_group group_oper_assoc by simp
  ultimately show a⁻¹·(b·a) = b by simp
  from A1 A2 show a·(b·a⁻¹) = b
     using inverse_in_group IsCommutative_def group0_2_L16
     by simp
qed
```

Another lemma about cancelling with two elements.

```
lemma (in group0) group0_4_L6AA:
  assumes A1: f {is commutative on} G and A2: a∈G  b∈G
  shows
  a·b⁻¹·a⁻¹ = b⁻¹
  using prems inverse_in_group group0_4_L6A
  by auto
```

Another lemma about cancelling with two elements.

```
lemma (in group0) group0_4_L6AB:
  assumes A1: f {is commutative on} G and A2: a∈G  b∈G
  shows
  a·(a·b)⁻¹ = b⁻¹
  a·(b·a⁻¹) = b
proof -
    from A2 have a·(a·b)⁻¹ = a·(b⁻¹·a⁻¹)
      using group_inv_of_two by simp
    also from A2 have ... = a·b⁻¹·a⁻¹
      using inverse_in_group group_oper_assoc by simp
    also from A1 A2 have ... =  b⁻¹
      using group0_4_L6AA by simp
    finally show a·(a·b)⁻¹ = b⁻¹ by simp
    from A1 A2 have a·(b·a⁻¹) = a·(a⁻¹·b)
      using inverse_in_group IsCommutative_def by simp
    also from A2 have ... = b
      using inverse_in_group group_oper_assoc group0_2_L6 group0_2_L2
      by simp
    finally show a·(b·a⁻¹) = b by simp
qed
```

Another lemma about cancelling with two elements.

```
lemma (in group0) group0_4_L6AC:
  assumes f {is commutative on} G and a∈G  b∈G
  shows a·(a·b⁻¹)⁻¹ = b
  using prems inverse_in_group group0_4_L6AB group_inv_of_inv
  by simp
```

In abelian groups we can cancel an element with its inverse even if separated by two other elements.

```
lemma (in group0) group0_4_L6B: assumes A1: f {is commutative on} G
```

```
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·c·a⁻¹ = b·c
  a⁻¹·b·c·a = b·c
proof -
   from A2 have
     a·b·c·a⁻¹ = a·(b·c)·a⁻¹
     a⁻¹·b·c·a = a⁻¹·(b·c)·a
    using group_op_closed group_oper_assoc inverse_in_group
    by auto
  with A1 A2 show
    a·b·c·a⁻¹ = b·c
    a⁻¹·b·c·a = b·c
    using group_op_closed group0_4_L6A
    by auto
qed
```

In abelian groups we can cancel an element with its inverse even if separated by three other elements.

```
lemma (in group0) group0_4_L6C: assumes A1: f {is commutative on} G
  and A2: a∈G b∈G c∈G d∈G
  shows a·b·c·d·a⁻¹ = b·c·d
proof -
  from A2 have a·b·c·d·a⁻¹ = a·(b·c·d)·a⁻¹
    using group_op_closed group_oper_assoc
    by simp
  with A1 A2 show thesis
    using group_op_closed group0_4_L6A
    by simp
qed
```

Another couple of useful rearrangements of three elements and cancelling.

```
lemma (in group0) group0_4_L6D:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b⁻¹·(a·c⁻¹)⁻¹ = c·b⁻¹
  (a·c)⁻¹·(b·c) = a⁻¹·b
  a·(b·(c·a⁻¹·b⁻¹)) = c
  a·b·c⁻¹·(c·a⁻¹) = b
proof -
  from A2 have T:
    a⁻¹ ∈ G  b⁻¹ ∈ G  c⁻¹ ∈ G
    a·b ∈ G  a·b⁻¹ ∈ G  c⁻¹·a⁻¹ ∈ G  c·a⁻¹ ∈ G
    using inverse_in_group group_op_closed by auto
  with A1 A2 show a·b⁻¹·(a·c⁻¹)⁻¹ = c·b⁻¹
    using group0_2_L12 group_oper_assoc group0_4_L6B
    IsCommutative_def by simp
  from A2 T have (a·c)⁻¹·(b·c) = c⁻¹·a⁻¹·b·c
```

```
    using group_inv_of_two group_oper_assoc by simp
  also from A1 A2 T have ... = a⁻¹·b
    using group0_4_L6B by simp
  finally show (a·c)⁻¹·(b·c) = a⁻¹·b
    by simp
  from A1 A2 T show a·(b·(c·a⁻¹·b⁻¹)) = c
    using group_oper_assoc group0_4_L6B group0_4_L6A
    by simp
  from T have a·b·c⁻¹·(c·a⁻¹) = a·b·(c⁻¹·(c·a⁻¹))
    using group_oper_assoc by simp
  also from A1 A2 T have ... = b
    using group_oper_assoc group0_2_L6 group0_2_L2 group0_4_L6A
    by simp
  finally show a·b·c⁻¹·(c·a⁻¹) = b by simp
qed
```

Another useful rearrangement of three elements and cancelling.

```
lemma (in group0) group0_4_L6E:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G
  shows
  a·b·(a·c)⁻¹ = b·c⁻¹
proof -
  from A2 have T: b⁻¹ ∈ G  c⁻¹ ∈ G
    using inverse_in_group by auto
  with A1 A2 have
    a·(b⁻¹)⁻¹·(a·(c⁻¹)⁻¹)⁻¹ = c⁻¹·(b⁻¹)⁻¹
    using group0_4_L6D by simp
  with A1 A2 T show a·b·(a·c)⁻¹ = b·c⁻¹
    using group_inv_of_inv IsCommutative_def
    by simp
qed
```

A rearrangement with two elements and canceelling, special case of group0_4_L6D when $c = b^{-1}$.

```
lemma (in group0) group0_4_L6F:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G
  shows a·b⁻¹·(a·b)⁻¹ = b⁻¹·b⁻¹
proof -
  from A2 have b⁻¹ ∈ G
    using inverse_in_group by simp
  with A1 A2 have a·b⁻¹·(a·(b⁻¹)⁻¹)⁻¹ = b⁻¹·b⁻¹
    using group0_4_L6D by simp
  with A2 show a·b⁻¹·(a·b)⁻¹ = b⁻¹·b⁻¹
    using group_inv_of_inv by simp
qed
```

Some other rearrangements with four elements. The algorithm for proof as in group0_4_L2 works very well here.

```
lemma (in group0) rearr_ab_gr_4_elemA:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b·c·d = a·d·b·c
  a·b·c·d = a·c·(b·d)
proof -
  from A1 A2 have a·b·c·d = d·(a·b·c)
    using  IsCommutative_def group_op_closed
    by simp
  also from A2 have ... = d·a·b·c
    using group_op_closed group_oper_assoc
    by simp
  also from A1 A2 have ... = a·d·b·c
    using IsCommutative_def group_op_closed
    by simp
  finally show a·b·c·d = a·d·b·c
    by simp
  from A1 A2 have a·b·c·d = c·(a·b)·d
    using IsCommutative_def group_op_closed
    by simp
  also from A2 have ... = c·a·b·d
    using group_op_closed group_oper_assoc
    by simp
  also from A1 A2 have ... = a·c·b·d
    using IsCommutative_def group_op_closed
    by simp
  also from A2 have ... = a·c·(b·d)
    using group_op_closed group_oper_assoc
    by simp
  finally show a·b·c·d = a·c·(b·d)
    by simp
qed
```

Some rearrangements with four elements and inverse that are applications of rearr_ab_gr_4_elem

```
lemma (in group0) rearr_ab_gr_4_elemB:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b⁻¹·c⁻¹·d⁻¹ = a·d⁻¹·b⁻¹·c⁻¹
  a·b·c·d⁻¹ = a·d⁻¹·b·c
  a·b·c⁻¹·d⁻¹ =  a·c⁻¹·(b·d⁻¹)
proof -
  from A2 have T: b⁻¹ ∈ G  c⁻¹ ∈ G  d⁻¹ ∈ G
    using inverse_in_group by auto
  with A1 A2 show
    a·b⁻¹·c⁻¹·d⁻¹ = a·d⁻¹·b⁻¹·c⁻¹
```

```
    a·b·c·d⁻¹ = a·d⁻¹·b·c
    a·b·c⁻¹·d⁻¹ =  a·c⁻¹·(b·d⁻¹)
    using rearr_ab_gr_4_elemA by auto
qed
```

Some rearrangement lemmas with four elements.

```
lemma (in group0) group0_4_L7:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b·c·d⁻¹ = a·d⁻¹· b·c
  a·d·(b·d·(c·d))⁻¹ = a·(b·c)⁻¹·d⁻¹
  a·(b·c)·d = a·b·d·c
proof -
  from A2 have T:
    b·c ∈ G d⁻¹ ∈ G b⁻¹∈G c⁻¹∈G
    d⁻¹·b ∈ G c⁻¹·d ∈ G (b·c)⁻¹ ∈ G
    b·d ∈ G  b·d·c ∈ G  (b·d·c)⁻¹ ∈ G
    a·d ∈ G  b·c ∈ G
    using group_op_closed inverse_in_group
    by auto
  with A1 A2 have a·b·c·d⁻¹ = a·(d⁻¹·b·c)
    using group_oper_assoc group0_4_L4A by simp
  also from A2 T have a·(d⁻¹·b·c) = a·d⁻¹·b·c
    using group_oper_assoc by simp
  finally show a·b·c·d⁻¹ = a·d⁻¹· b·c by simp
  from A2 T have a·d·(b·d·(c·d))⁻¹ = a·d·(d⁻¹·(b·d·c)⁻¹)
    using group_oper_assoc group_inv_of_two by simp
  also from A2 T have ... = a·(b·d·c)⁻¹
    using group_oper_assoc group0_2_L16 by simp
  also from A1 A2 have ... =  a·(d·(b·c))⁻¹
    using IsCommutative_def group_oper_assoc by simp
  also from A2 T have ... = a·((b·c)⁻¹·d⁻¹)
    using group_inv_of_two by simp
  also from A2 T have ... =  a·(b·c)⁻¹·d⁻¹
    using group_oper_assoc by simp
  finally show a·d·(b·d·(c·d))⁻¹ = a·(b·c)⁻¹·d⁻¹
    by simp
  from A2 have a·(b·c)·d = a·(b·(c·d))
    using group_op_closed group_oper_assoc by simp
  also from A1 A2 have ... =  a·(b·(d·c))
    using IsCommutative_def group_op_closed by simp
  also from A2 have ... =  a·b·d·c
    using group_op_closed group_oper_assoc by simp
  finally show a·(b·c)·d = a·b·d·c by simp
qed
```

Some other rearrangements with four elements.

```
lemma (in group0) group0_4_L8:
```

```
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·(b·c)⁻¹ = (a·d⁻¹·c⁻¹)·(d·b⁻¹)
  a·b·(c·d) = c·a·(b·d)
  a·b·(c·d) = a·c·(b·d)
  a·(b·c⁻¹)·d = a·b·d·c⁻¹
  (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = a·c⁻¹
proof -
  from A2 have T:
    b·c ∈ G a·b ∈ G d⁻¹ ∈ G b⁻¹∈G c⁻¹∈G
    d⁻¹·b ∈ G c⁻¹·d ∈ G (b·c)⁻¹ ∈ G
    a·b ∈ G  (c·d)⁻¹ ∈ G  (b·d⁻¹)⁻¹ ∈ G  d·b⁻¹ ∈ G
    using group_op_closed inverse_in_group
    by auto
  from A2 have a·(b·c)⁻¹ = a·c⁻¹·b⁻¹ using group0_2_L14A by blast
  moreover from A2 have a·c⁻¹ = (a·d⁻¹)·(d·c⁻¹) using group0_2_L14A
    by blast
  ultimately have a·(b·c)⁻¹ = (a·d⁻¹)·(d·c⁻¹)·b⁻¹ by simp
  with A1 A2 T have a·(b·c)⁻¹= a·d⁻¹·(c⁻¹·d)·b⁻¹
    using IsCommutative_def by simp
  with A2 T show a·(b·c)⁻¹ = (a·d⁻¹·c⁻¹)·(d·b⁻¹)
    using group_op_closed group_oper_assoc by simp
  from A2 T have a·b·(c·d) = a·b·c·d
    using group_oper_assoc by simp
  also have a·b·c·d = c·a·b·d
  proof -
    from A1 A2 have a·b·c·d = c·(a·b)·d
      using IsCommutative_def group_op_closed
      by simp
    also from A2 have ... = c·a·b·d
      using group_op_closed group_oper_assoc
      by simp
    finally show thesis by simp
  qed
  also from A2 have c·a·b·d =  c·a·(b·d)
    using group_op_closed group_oper_assoc
    by simp
  finally show a·b·(c·d) = c·a·(b·d) by simp
  with A1 A2 show a·b·(c·d) = a·c·(b·d)
    using IsCommutative_def by simp
  from A1 A2 T show a·(b·c⁻¹)·d = a·b·d·c⁻¹
    using group0_4_L7 by simp
  from T have (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = (a·b)·((c·d)⁻¹·(b·d⁻¹)⁻¹)
    using group_oper_assoc by simp
  also from A1 A2 T have ... = (a·b)·(c⁻¹·d⁻¹·(d·b⁻¹))
    using group_inv_of_two group0_2_L12 IsCommutative_def
    by simp
  also from T have ... = (a·b)·(c⁻¹·(d⁻¹·(d·b⁻¹)))
```

```
    using group_oper_assoc by simp
  also from A1 A2 T have ... = a·c⁻¹
    using group_oper_assoc group0_2_L6 group0_2_L2 IsCommutative_def
    group0_2_L16 by simp
  finally show (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = a·c⁻¹
    by simp
qed
```

Some other rearrangements with four elements.

```
lemma (in group0) group0_4_L8A:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  shows
  a·b⁻¹·(c·d⁻¹) = a·c·(b⁻¹·d⁻¹)
  a·b⁻¹·(c·d⁻¹) = a·c·b⁻¹·d⁻¹
proof -
  from A2 have
    T: a∈G  b⁻¹ ∈ G  c∈G  d⁻¹ ∈ G
    using inverse_in_group by auto
  with A1 show a·b⁻¹·(c·d⁻¹) = a·c·(b⁻¹·d⁻¹)
    by (rule group0_4_L8)
  with A2 T show  a·b⁻¹·(c·d⁻¹) = a·c·b⁻¹·d⁻¹
    using group_op_closed group_oper_assoc
    by simp
qed
```

Another rearrangement about equation.

```
lemma (in group0) group0_4_L9:
  assumes A1: f {is commutative on} G
  and A2: a∈G  b∈G  c∈G  d∈G
  and A3: a = b·c⁻¹·d⁻¹
  shows
  d = b·a⁻¹·c⁻¹
  d = a⁻¹·b·c⁻¹
  b = a·d·c
proof -
  from A2 have T:
    a⁻¹ ∈ G  c⁻¹ ∈ G  d⁻¹ ∈ G  b·c⁻¹ ∈ G
    using group_op_closed inverse_in_group
    by auto
  with A2 A3 have a·(d⁻¹)⁻¹ =  b·c⁻¹
    using group0_2_L18 by simp
  with A2 have b·c⁻¹ = a·d
    using group_inv_of_inv by simp
  with A2 T have I: a⁻¹·(b·c⁻¹) = d
    using group0_2_L18 by simp
  with A1 A2 T show
    d = b·a⁻¹·c⁻¹
    d = a⁻¹·b·c⁻¹
```

```
    using group_oper_assoc IsCommutative_def by auto
  from A3 have a·d·c = (b·c⁻¹·d⁻¹)·d·c by simp
  also from A2 T have ... = b·c⁻¹·(d⁻¹·d)·c
    using group_oper_assoc by simp
  also from A2 T have ... =  b·c⁻¹·c
    using group0_2_L6 group0_2_L2 by simp
  also from A2 T have ... =  b·(c⁻¹·c)
    using group_oper_assoc by simp
  also from A2 have ... = b
    using group0_2_L6 group0_2_L2 by simp
  finally have a·d·c = b by simp
  thus b = a·d·c by simp
qed
```

13.5 Translations

In this section we consider translations. Translations are maps $T : G \rightarrow G$ of the form $T_g(a) = g \cdot a$ or $T_g(a) = a \cdot g$. We also consider two-dimensional translations $T_g : G \times G \rightarrow G \times G$, where $T_g(a, b) = (a \cdot g, b \cdot g)$ or$T_g(a, b) = (g \cdot a, g \cdot b)$.

```
constdefs
  RightTranslation(G,P,g) ≡ {<a,b> ∈ G×G. P<a,g> = b}

  LeftTranslation(G,P,g) ≡ {<a,b> ∈ G×G. P<g,a> = b}

  RightTranslation2(G,P,g) ≡
  {<x,y> ∈ (G×G)×(G×G). ⟨P<fst(x),g>, P<snd(x),g>⟩ = y}

  LeftTranslation2(G,P,g) ≡
  {<x,y> ∈ (G×G)×(G×G). ⟨P<g,fst(x)>, P<g,snd(x)>⟩ = y}
```

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,f,g) : G→G
  LeftTranslation(G,f,g) : G→G
  RightTranslation2(G,f,g) : (G×G)→(G×G)
  LeftTranslation2(G,f,g) : (G×G)→(G×G)
proof -
  from A1 have ∀a∈G. a·g ∈ G ∀a∈G. g·a ∈ G
    ∀x ∈ G×G.  <fst(x)·g, snd(x)·g> ∈ G×G
    ∀x ∈ G×G.  <g·fst(x),g·snd(x)> ∈ G×G
    using group_oper_assocA apply_funtype by auto
  then show RightTranslation(G,f,g) : G→G
    LeftTranslation(G,f,g) : G→G
    RightTranslation2(G,f,g) : (G×G)→(G×G)
    LeftTranslation2(G,f,g) : (G×G)→(G×G)
    using RightTranslation_def LeftTranslation_def
```

```
    RightTranslation2_def LeftTranslation2_def func1_1_L11A
  by auto
qed
```

The values of the translations are what we expect.

```
lemma (in group0) group0_5_L2: assumes A1: g∈G a∈G
  shows
  RightTranslation(G,f,g)(a) = a·g
  LeftTranslation(G,f,g)(a) = g·a
  using prems group0_5_L1 RightTranslation_def LeftTranslation_def
    func1_1_L11B by auto
```

The values of the two-dimensional translations are what we expect.

```
lemma (in group0) group0_5_L3: assumes A1: g∈G a∈G b∈G
  shows RightTranslation2(G,f,g)<a,b> = <a·g,b·g>
  LeftTranslation2(G,f,g)<a,b> = <g·a,g·b>
  using prems RightTranslation2_def LeftTranslation2_def
    group0_5_L1 func1_1_L11B by auto
```

Composition of left translations is a left translation by the product.

```
lemma (in group0) group0_5_L4: assumes A1:g∈G h∈G a∈G
  and A2: T_g = LeftTranslation(G,f,g) T_h = LeftTranslation(G,f,h)
  shows T_g(T_h(a)) = g·h·a
  T_g(T_h(a)) = LeftTranslation(G,f,g·h)(a)
proof -
  from A1 have T1:h·a∈G g·h∈G
    using group_oper_assocA apply_funtype by auto
  with A1 A2 show T_g(T_h(a)) = g·h·a
    using group0_5_L2 group_oper_assoc by simp
  with A1 A2 T1 show
    T_g(T_h(a)) = LeftTranslation(G,f,g·h)(a)
    using group0_5_L2 group_oper_assoc by simp
qed
```

Composition of right translations is a right translation by the product.

```
lemma (in group0) group0_5_L5: assumes A1:g∈G h∈G a∈G
  and A2: T_g = RightTranslation(G,f,g) T_h = RightTranslation(G,f,h)
  shows T_g(T_h(a)) = a·h·g
  T_g(T_h(a)) = RightTranslation(G,f,h·g)(a)
proof -
  from A1 have T1: a·h∈G h·g ∈G
    using group_oper_assocA apply_funtype by auto
  with A1 A2 show T_g(T_h(a)) = a·h·g
    using group0_5_L2 group_oper_assoc by simp
  with A1 A2 T1 show
    T_g(T_h(a)) = RightTranslation(G,f,h·g)(a)
    using group0_5_L2 group_oper_assoc by simp
qed
```

The image of a set under a composition of translations is the same as the image under translation by a product.

```
lemma (in group0) group0_5_L6: assumes A1: g∈G h∈G and A2: A⊆G
  and A3: T_g = RightTranslation(G,f,g) T_h = RightTranslation(G,f,h)
  shows T_g(T_h(A)) = {a·h·g. a∈A}
proof -
  from A2 have T1:∀a∈A. a∈G by auto
  from A1 A3 have T_g : G→G T_h : G→G
    using group0_5_L1 by auto
  with A1 A2 T1 A3 show T_g(T_h(A)) = {a·h·g. a∈A}
    using func1_1_L15C group0_5_L5 by simp
qed
```

13.6 Odd functions

This section is about odd functions.

Odd functions are those that commute with the group inverse: $f(a^{-1}) = (f(a))^{-1}$.

```
constdefs

  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,f,p) ←→ (∀a∈G. p(a⁻¹) = (p(a))⁻¹)
  using IsOdd_def by simp
```

We can express the definition of an odd function in two ways.

```
lemma (in group0) group0_6_L2:
  assumes A1: p : G→G shows
  (∀a∈G. p(a⁻¹) = (p(a))⁻¹) ←→ (∀a∈G. (p(a⁻¹))⁻¹ = p(a))
proof
  assume ∀a∈G. p(a⁻¹) = (p(a))⁻¹
  with A1 show ∀a∈G. (p(a⁻¹))⁻¹ = p(a)
    using apply_funtype group_inv_of_inv by simp
next assume A2: ∀a∈G. (p(a⁻¹))⁻¹ = p(a)
  { fix a assume a∈G
    with A1 A2 have p(a⁻¹) ∈ G   ((p(a⁻¹))⁻¹)⁻¹ =  (p(a))⁻¹
      using apply_funtype inverse_in_group by auto
    then have p(a⁻¹) = (p(a))⁻¹
      using group_inv_of_inv by simp
  } then show ∀a∈G. p(a⁻¹) = (p(a))⁻¹ by simp
qed

end
```

14 Group_ZF_1.thy

theory `Group_ZF_1` **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 $e \cdot a = a$ and $a \cdot e = a$. 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 neutral elements e is not empty. One way around this is to first use condition A to define the notion of monoid, then prove the uniqueness of e and then use the condition B to define groups. However, there 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 assocative 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 romantic proofs come from an Aug. 14, 2005, 2006 post by buli on the matematyka.org forum.

14.1 An alternative definition of group

We will use the multiplicative notation for the group. To do this, we define a context (locale) similar to group0, 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>
```

A set G with an associative operation that satisfies condition C is a group, as defined in `Group_ZF` theory file.

```
theorem (in group2) Group_ZF_1_T1:
  assumes A1: G≠0 and A2: P {is associative on} G
  and A3: ∀a∈G.∀b∈G. ∃x∈G. a·x = b
  and A4: ∀a∈G.∀b∈G. ∃y∈G. y·a = b
```

```
  shows IsAgroup(G,P)
proof -
  from A1 obtain a where D1: a∈G by auto
  with A3 obtain x where D2: x∈G and D3: a·x = a
    by auto
  from D1 A4 obtain y where D4: y∈G and D5: y·a = a
    by auto
  have T1: ∀b∈G. b = b·x ∧ b = y·b
  proof
    fix b assume A5: b∈G
     with D1 A4 obtain y_b where D6: y_b∈G
      and D7: y_b·a = b by auto
    from A5 D1 A3 obtain x_b where D8: x_b∈G
      and D9: a·x_b = b by auto
    from D7 D3 D9 D5 have
      b = y_b·(a·x)  b = (y·a)·x_b by auto
    moreover from D1 D2 D4 D8 D6 A2 have
      (y·a)·x_b = y·(a·x_b)  y_b·(a·x) = (y_b·a)·x
      using IsAssociative_def by auto
    moreover from D7 D9 have
      (y_b·a)·x = b·x  y·(a·x_b) = y·b
      by auto
    ultimately show b = b·x ∧ b = y·b by simp
  qed
  moreover have x = y
  proof -
    from D2 T1 have x = y·x by simp
    also from D4 T1 have y·x = y by simp
    finally show thesis by simp
  qed
  ultimately have ∀b∈G. b·x = b ∧ x·b = b by simp
  with D2 A2 have IsAmonoid(G,P) using IsAmonoid_def by auto
  with A3 show IsAgroup(G,P)
    using monoid0_def monoid0.group0_1_L3 IsAgroup_def
    by simp
qed

end
```

15 Group_ZF_2.thy

theory Group_ZF_2 **imports** Group_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.

15.1 Lifting groups to function spaces

If we have a monoid (group) G than we get a monoid (group) structure on a space of functions valued in in G by defining $(f \cdot g)(x) := f(x) \cdot g(x)$. We call this process "lifting the monoid (group) to function space. This section formalizes this "lifting".

The lifted operation is an operation on the function space.

```
lemma (in monoid0) Group_ZF_2_1_L0A:
  assumes A1: F = f {lifted to function space over} X
  shows F : (X→G)×(X→G)→(X→G)
proof -
  from monoidAsssum have f : G×G→G
    using IsAmonoid_def IsAssociative_def by simp
  with A1 show thesis
    using func_ZF_1_L3 group0_1_L3B by auto
qed
```

The result of the lifted operation is in the function space.

```
lemma (in monoid0) Group_ZF_2_1_L0:
  assumes A1:F = f {lifted to function space over} X
  and A2:s:X→G r:X→G
  shows F<s,r> : X→G
proof -
  from A1 have F : (X→G)×(X→G)→(X→G)
    using Group_ZF_2_1_L0A
    by simp
  with A2 show thesis using apply_funtype
    by simp
qed
```

The lifted monoid operation has a neutral element, namely the constant function with the neutral element as the value.

```
lemma (in monoid0) Group_ZF_2_1_L1:
  assumes A1: F = f {lifted to function space over} X
  and A2: E = ConstantFunction(X,TheNeutralElement(G,f))
  shows E : X→G ∧ (∀s∈X→G. F<E,s> = s ∧ F<s,E> = s)
proof
```

```
  from A2 show T1:E : X→G
    using group0_1_L3 func1_3_L1 by simp
  show ∀s∈X→G. F<E,s> = s ∧ F<s,E> = s
  proof
    fix s assume A3:s:X→G
    from monoidAsssum have T2:f : G×G→G
      using IsAmonoid_def IsAssociative_def by simp
    from A3 A1 T1 have
      F<E,s> : X→G F<s,E> : X→G s : X→G
      using Group_ZF_2_1_L0 by auto
    moreover from T2 A1 T1 A2 A3 have
      ∀x∈X. (F<E,s>)(x) = s(x)
      ∀x∈X. (F<s,E>)(x) = s(x)
      using func_ZF_1_L4 group0_1_L3B func1_3_L2
        apply_type group0_1_L3 by auto
    ultimately show
      F<E,s> = s ∧ F<s,E> = s
      using fun_extension_iff by auto
  qed
qed
```

Monoids can be lifted to a function space.

```
lemma (in monoid0) Group_ZF_2_1_T1:
  assumes A1:F = f {lifted to function space over} X
  shows IsAmonoid(X→G,F)
proof -
  from monoidAsssum A1 have
    F {is associative on} (X→G)
    using IsAmonoid_def func_ZF_2_L4 group0_1_L3B
    by auto
  moreover from A1 have
    ∃ E ∈ X→G. ∀s ∈ X→G. F<E,s> = s ∧ F<s,E> = s
    using Group_ZF_2_1_L1 by blast
  ultimately show thesis using IsAmonoid_def
    by simp
qed
```

The constant function with the neutral element as the value is the neutral element of the lifted monoid.

```
lemma Group_ZF_2_1_L2:
  assumes A1:IsAmonoid(G,f)
  and A2:F = f {lifted to function space over} X
  and A3:E = ConstantFunction(X,TheNeutralElement(G,f))
  shows E = TheNeutralElement(X→G,F)
proof -
  from A1 A2 have
     T1:monoid0(G,f) and T2:monoid0(X→G,F)
    using monoid0_def monoid0.Group_ZF_2_1_T1
    by auto
```

```
  from T1 A2 A3 have
    E : X→G ∧ (∀s∈X→G. F<E,s> = s ∧ F<s,E> = s)
    using monoid0.Group_ZF_2_1_L1 by simp
  with T2 show thesis
    using monoid0.group0_1_L4 by auto
qed
```

The lifted operation acts on the functions in a natural way defined by the group operation.

```
lemma (in group0) Group_ZF_2_1_L3:
  assumes A1:F = f {lifted to function space over} X
  and A2:s:X→G r:X→G
  and A3:x∈X
  shows (F<s,r>)(x) = s(x)·r(x)
proof -
  from groupAssum A1 A2 A3 show thesis
     using IsAgroup_def IsAmonoid_def IsAssociative_def
      group0_2_L1 monoid0.group0_1_L3B func_ZF_1_L4
     by auto
qed
```

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 = f {lifted to function space over} X
  shows monoid0(X→G,F)
proof -
  from A1 show thesis
    using group0_2_L1 monoid0.Group_ZF_2_1_T1 monoid0_def
    by simp
qed
```

The compostion of a function $f : X \to G$ with the group inverse is a right inverse for the lifted group. Recall that in the group0 context e is the neutral element of the group.

```
lemma (in group0) Group_ZF_2_1_L5:
  assumes A1: F = f {lifted to function space over} X
  and A2: s : X→G
  and A3: i = GroupInv(G,f) O s
  shows i: X→G F<s,i> = TheNeutralElement(X→G,F)
proof -
  let E = ConstantFunction(X,1)
  have E : X→G
    using group0_2_L2 func1_3_L1 by simp
  moreover from groupAssum A2 A3 A1 have
    F<s,i> :  X→G using group0_2_T2 comp_fun
      Group_ZF_2_1_L4 monoid0.group0_1_L1
    by simp
```

```
  moreover from groupAssum A2 A3 A1 have
    ∀x∈X. (F<s,i>)(x) = E(x)
    using group0_2_T2 comp_fun Group_ZF_2_1_L3
      comp_fun_apply apply_funtype group0_2_L6 func1_3_L2
    by simp
  moreover from groupAssum A1 have
    E = TheNeutralElement(X→G,F)
    using IsAgroup_def Group_ZF_2_1_L2 by simp
  ultimately show F<s,i> = TheNeutralElement(X→G,F)
    using fun_extension_iff IsAgroup_def Group_ZF_2_1_L2
    by simp
  from groupAssum A2 A3 show i: X→G
    using group0_2_T2 comp_fun by simp
qed
```

Groups can be lifted to the function space.

```
theorem (in group0) Group_ZF_2_1_T2:
  assumes A1: F = f {lifted to function space over} X
  shows IsAgroup(X→G,F)
proof -
  from A1 have IsAmonoid(X→G,F)
    using group0_2_L1 monoid0.Group_ZF_2_1_T1
    by simp
  moreover have
    ∀s∈X→G. ∃i∈X→G. F<s,i> = TheNeutralElement(X→G,F)
  proof
    fix s assume A2: s : X→G
    let i = GroupInv(G,f) O s
    from groupAssum A2 have i:X→G
      using group0_2_T2 comp_fun by simp
    moreover from A1 A2 have
      F<s,i> = TheNeutralElement(X→G,F)
      using Group_ZF_2_1_L5 by fast
   ultimately show ∃i∈X→G. F<s,i> = TheNeutralElement(X→G,F)
      by auto
  qed
  ultimately show thesis using IsAgroup_def
    by simp
qed
```

What is the group inverse for the lifted group?

```
lemma (in group0) Group_ZF_2_1_L6:
  assumes A1: F = f {lifted to function space over} X
  shows ∀s∈(X→G). GroupInv(X→G,F)(s) = GroupInv(G,f) O s
proof -
  from A1 have  group0(X→G,F)
    using group0_def Group_ZF_2_1_T2
    by simp
  moreover from A1 have ∀s∈X→G. GroupInv(G,f) O s : X→G ∧
```

```
    F<s,GroupInv(G,f) O s> = TheNeutralElement(X→G,F)
    using Group_ZF_2_1_L5 by simp
  ultimately have
    ∀s∈(X→G).  GroupInv(G,f) O s = GroupInv(X→G,F)(s)
    by (rule group0.group0_2_L9A)
  thus thesis by simp
qed
```

What is the group inverse in a subgroup of the lifted group?

```
lemma (in group0) Group_ZF_2_1_L6A:
  assumes A1: F = f {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,f) O s
proof -
  from A1 have T1: group0(X→G,F)
    using group0_def Group_ZF_2_1_T2
    by simp
  with A2 A3 A4 have GroupInv(H,g)(s) = GroupInv(X→G,F)(s)
    using group0.group0_3_T1 restrict by simp
  moreover from T1 A1 A2 A4 have
    GroupInv(X→G,F)(s) = GroupInv(G,f) O s
    using group0.group0_3_L2 Group_ZF_2_1_L6 by blast
  ultimately show thesis by simp
qed
```

If a group is abelian, then its lift to a function space is also abelian.

```
lemma (in group0) Group_ZF_2_1_L7:
  assumes A1: F = f {lifted to function space over} X
  and A2: f {is commutative on} G
  shows F {is commutative on} (X→G)
proof-
  from A1 A2  have
    F {is commutative on} (X→range(f))
    using group_oper_assocA func_ZF_2_L2
    by simp
  moreover from groupAssum have range(f) = G
    using group0_2_L1 monoid0.group0_1_L3B
    by simp
  ultimately show thesis by simp
qed
```

15.2 Equivalence relations on groups

The goal of this section is to establish that (under some conditions) given an equivalence relation on a group or (monoid)we can project the group (monoid) structure on the quotient and obtain another group.

The neutral element class is neutral in the projection.

```
lemma (in monoid0) Group_ZF_2_2_L1:
  assumes A1: equiv(G,r) and A2:Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  and A4: e = TheNeutralElement(G,f)
  shows r{e} ∈ G//r ∧
  (∀c ∈ G//r. F<r{e},c> = c ∧  F<c,r{e}> = c)
proof
  from A4 show T1:r{e} ∈ G//r
    using group0_1_L3 quotientI
    by simp
  show
    ∀c ∈ G//r. F<r{e},c> = c ∧  F<c,r{e}> = c
  proof
    fix c assume A5:c ∈ G//r
    then obtain g where D1:g∈G c = r{g}
      using quotient_def by auto
    with A1 A2 A3 A4 D1 show
      F<r{e},c> = c ∧  F<c,r{e}> = c
      using group0_1_L3 EquivClass_1_L10 group0_1_L3
      by simp
  qed
qed
```

The projected structure is a monoid.

```
theorem (in monoid0) Group_ZF_2_2_T1:
  assumes A1: equiv(G,r) and A2: Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  shows IsAmonoid(G//r,F)
proof -
  let E = r{TheNeutralElement(G,f)}
  from A1 A2 A3 have
    E ∈ G//r ∧ (∀c∈G//r. F<E,c> = c ∧ F<c,E> = c)
    using Group_ZF_2_2_L1 by simp
  hence
    ∃E∈G//r. ∀ c∈G//r. F<E,c> = c ∧ F<c,E> = c
    by auto
  with monoidAsssum A1 A2 A3 show thesis
    using IsAmonoid_def EquivClass_2_T2
    by simp
qed
```

The class of the neutral element is the neutral element of the projected monoid.

```
lemma Group_ZF_2_2_L1:
  assumes A1: IsAmonoid(G,f)
  and A2: equiv(G,r) and A3: Congruent2(r,f)
  and A4: F = ProjFun2(G,r,f)
```

```
  and A5: e = TheNeutralElement(G,f)
  shows  r{e} = TheNeutralElement(G//r,F)
proof -
  from A1 A2 A3 A4 have
    T1:monoid0(G,f) and T2:monoid0(G//r,F)
    using monoid0_def monoid0.Group_ZF_2_2_T1 by auto
  from T1 A2 A3 A4 A5 have r{e} ∈ G//r ∧
    (∀c ∈ G//r. F<r{e},c> = c ∧  F<c,r{e}> = c)
    using monoid0.Group_ZF_2_2_L1 by simp
  with T2 show thesis using monoid0.group0_1_L4
    by auto
qed
```

The projected operation can be defined in terms of the group operation on representants in a natural way.

```
lemma (in group0) Group_ZF_2_2_L2:
  assumes A1: equiv(G,r) and A2: Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  and A4: a∈G b∈G
  shows F<r{a},r{b}> = r{a·b}
proof -
  from A1 A2 A3 A4 show thesis
    using EquivClass_1_L10 by simp
qed
```

The class of the inverse is a right inverse of the class.

```
lemma (in group0) Group_ZF_2_2_L3:
  assumes A1: equiv(G,r) and A2: Congruent2(r,f)
  and A3: F = ProjFun2(G,r,f)
  and A4: a∈G
  shows F⟨r{a},r{a⁻¹}⟩ = TheNeutralElement(G//r,F)
proof -
  from A1 A2 A3 A4 have
    F⟨r{a},r{a⁻¹}⟩ = r{1}
    using inverse_in_group Group_ZF_2_2_L2 group0_2_L6
    by simp
  with groupAssum A1 A2 A3 show thesis
    using IsAgroup_def Group_ZF_2_2_L1 by simp
qed
```

The group structure can be projected to the quotient space.

```
theorem (in group0) Group_ZF_3_T2:
  assumes A1: equiv(G,r) and A2: Congruent2(r,f)
  shows IsAgroup(G//r,ProjFun2(G,r,f))
proof -
  let F = ProjFun2(G,r,f)
  let E = TheNeutralElement(G//r,F)
  from groupAssum A1 A2 have IsAmonoid(G//r,F)
    using IsAgroup_def monoid0_def monoid0.Group_ZF_2_2_T1
```

```
    by simp
  moreover have
    ∀c∈G//r. ∃b∈G//r. F<c,b> = E
  proof
    fix c assume A3: c ∈ G//r
    then obtain g where D1: g∈G  c = r{g}
      using quotient_def by auto
    let b = r{g⁻¹}
    from D1 have b ∈ G//r
      using inverse_in_group quotientI
      by simp
    moreover from A1 A2 D1 have
      F<c,b> = E
      using Group_ZF_2_2_L3 by simp
    ultimately show ∃b∈G//r. F<c,b> = E
      by auto
  qed
  ultimately show thesis
    using IsAgroup_def by simp
qed
```

The group inverse (in the projected group) of a class is the class of the inverse.

```
lemma (in group0) Group_ZF_2_2_L4:
  assumes A1: equiv(G,r) and
  A2: Congruent2(r,f) and
  A3: F = ProjFun2(G,r,f) and
  A4: a∈G
  shows r{a⁻¹} = GroupInv(G//r,F)(r{a})
proof -
  from A1 A2 A3 have group0(G//r,F)
    using Group_ZF_3_T2 group0_def by simp
  moreover from A4 have
    r{a} ∈ G//r  r{a⁻¹} ∈ G//r
    using inverse_in_group quotientI by auto
  moreover from A1 A2 A3 A4 have
    F⟨r{a},r{a⁻¹}⟩ = TheNeutralElement(G//r,F)
    using Group_ZF_2_2_L3 by simp
  ultimately show thesis
    by (rule group0.group0_2_L9)
qed
```

15.3 Normal subgroups and quotient groups

A normal subgrup N of a group G is such that aba^{-1} belongs to N if $a \in G, b \in N$. 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.

```
constdefs
  IsAnormalSubgroup(G,f,N) ≡ IsAsubgroup(N,f) ∧
  (∀n∈N.∀g∈G. f< f< g,n >,GroupInv(G,f)(g) > ∈ N)

  QuotientGroupRel(G,f,H) ≡
  {<a,b> ∈ G×G. f<a, GroupInv(G,f)(b)> ∈ H}

  QuotientGroupOp(G,f,H) ≡ ProjFun2(G,QuotientGroupRel(G,f,H ),f)
```

Definition of a normal subgroup in a more readable notation.

```
lemma (in group0) Group_ZF_2_4_L0:
  assumes IsAnormalSubgroup(G,f,H)
  and g∈G n∈H
  shows g·n·g⁻¹ ∈ H
  using prems IsAnormalSubgroup_def by simp
```

The quotient group relation is reflexive.

```
lemma (in group0) Group_ZF_2_4_L1:
  assumes IsAsubgroup(H,f)
  shows refl(G,QuotientGroupRel(G,f,H))
  using prems  group0_2_L6 group0_3_L5
    QuotientGroupRel_def refl_def by simp
```

The quotient group relation is symmetric.

```
lemma (in group0) Group_ZF_2_4_L2:
  assumes A1:IsAsubgroup(H,f)
  shows sym(QuotientGroupRel(G,f,H))
proof -
  {
    fix a b assume A2: <a,b> ∈ QuotientGroupRel(G,f,H)
    with A1 have (a·b⁻¹)⁻¹ ∈ H
      using QuotientGroupRel_def group0_3_T3A
      by simp
    moreover from A2 have (a·b⁻¹)⁻¹ =  b·a⁻¹
      using QuotientGroupRel_def group0_2_L12
      by simp
    ultimately have b·a⁻¹ ∈ H by simp
    with A2 have <b,a> ∈ QuotientGroupRel(G,f,H)
      using QuotientGroupRel_def by simp
  }
  then show thesis using symI by simp
qed
```

The quotient group relation is transistive.

```
lemma (in group0) Group_ZF_2_4_L3A:
  assumes A1: IsAsubgroup(H,f) and
  A2: <a,b> ∈ QuotientGroupRel(G,f,H) and
  A3: <b,c> ∈ QuotientGroupRel(G,f,H)
```

```
  shows <a,c> ∈ QuotientGroupRel(G,f,H)
proof -
  let r = QuotientGroupRel(G,f,H)
  from A2 A3 have T1:a∈G b∈G c∈G
    using QuotientGroupRel_def by auto
  from A1 A2 A3 have (a·b⁻¹)·(b·c⁻¹) ∈ H
    using  QuotientGroupRel_def group0_3_L6
    by simp
  moreover from T1 have
    a·c⁻¹ = (a·b⁻¹)·(b·c⁻¹)
    using group0_2_L14A by blast
  ultimately have a·c⁻¹ ∈ H
    by simp
  with T1 show thesis using QuotientGroupRel_def
    by simp
qed
```

The quotient group relation is an equivalence relation. Note we do not need the subgroup to be normal for this to be true.

```
lemma (in group0) Group_ZF_2_4_L3: assumes A1:IsAsubgroup(H,f)
  shows equiv(G,QuotientGroupRel(G,f,H))
proof -
  let r = QuotientGroupRel(G,f,H)
  from A1 have
     ∀a b c. (⟨a, b⟩ ∈ r  ∧  ⟨b, c⟩ ∈ r ⟶ ⟨a, c⟩ ∈ r)
    using Group_ZF_2_4_L3A by blast
  then have trans(r)
    using Fol1_L2 by blast
  with A1 show thesis
    using Group_ZF_2_4_L1 Group_ZF_2_4_L2
      QuotientGroupRel_def equiv_def
    by auto
qed
```

The next lemma states the essential condition for congruency of the group operation with respect to the quotient group relation.

```
lemma (in group0) Group_ZF_2_4_L4:
  assumes A1:IsAnormalSubgroup(G,f,H)
  and A2:⟨a1,a2⟩ ∈ QuotientGroupRel(G,f,H)
  and A3:⟨b1,b2⟩ ∈ QuotientGroupRel(G,f,H)
  shows ⟨a1·b1, a2·b2⟩ ∈ QuotientGroupRel(G,f,H)
proof -
  from A2 A3 have T1:
    a1∈G  a2∈G  b1∈G  b2∈G
    a1·b1 ∈ G  a2·b2 ∈ G
    b1·b2⁻¹ ∈ H  a1·a2⁻¹ ∈ H
    using QuotientGroupRel_def group0_2_L1 monoid0.group0_1_L1
    by auto
  with A1 show thesis using
```

```
    IsAnormalSubgroup_def group0_3_L6 group0_2_L15
    QuotientGroupRel_def by simp
qed
```

If the subgroup is normal, the group operation is congruent with respect to the quotient group relation.

```
lemma Group_ZF_2_4_L5A:
  assumes IsAgroup(G,f)
  and IsAnormalSubgroup(G,f,H)
  shows Congruent2(QuotientGroupRel(G,f,H),f)
  using prems group0_def group0.Group_ZF_2_4_L4 Congruent2_def
  by simp
```

The quotient group is indeed a group.

```
theorem Group_ZF_2_4_T1:
  assumes IsAgroup(G,f) and IsAnormalSubgroup(G,f,H)
  shows
  IsAgroup(G//QuotientGroupRel(G,f,H),QuotientGroupOp(G,f,H))
  using prems group0_def group0.Group_ZF_2_4_L3 IsAnormalSubgroup_def
    Group_ZF_2_4_L5A group0.Group_ZF_3_T2 QuotientGroupOp_def
  by simp
```

The class (coset)of the neutral element is the neutral element of the quotient group.

```
lemma Group_ZF_2_4_L5B:
  assumes IsAgroup(G,f) and IsAnormalSubgroup(G,f,H)
  and r = QuotientGroupRel(G,f,H)
  and e = TheNeutralElement(G,f)
  shows  r{e} = TheNeutralElement(G//r,QuotientGroupOp(G,f,H))
  using prems IsAnormalSubgroup_def group0_def
    IsAgroup_def group0.Group_ZF_2_4_L3 Group_ZF_2_4_L5A
    QuotientGroupOp_def Group_ZF_2_2_L1
  by simp
```

A group element is equivalent to the neutral element iff it is in the subgroup we divide the group by.

```
lemma (in group0) Group_ZF_2_4_L5C: assumes a∈G
  shows ⟨a,1⟩ ∈ QuotientGroupRel(G,f,H) ⟷ a∈H
  using prems QuotientGroupRel_def group_inv_of_one group0_2_L2
  by auto
```

A group element is in H iff its class is the neutral element of G/H.

```
lemma (in group0) Group_ZF_2_4_L5D:
  assumes A1: IsAnormalSubgroup(G,f,H) and
  A2: a∈G and
  A3: r = QuotientGroupRel(G,f,H) and
  A4: TheNeutralElement(G//r,QuotientGroupOp(G,f,H)) = e
  shows r{a} = e ⟷ ⟨a,1⟩ ∈ r
```

```
proof
  assume r{a} = e
  with groupAssum prems have
    r{1} = r{a} and I: equiv(G,r)
    using Group_ZF_2_4_L5B IsAnormalSubgroup_def Group_ZF_2_4_L3
    by auto
  with A2 have ⟨1,a⟩ ∈ r using eq_equiv_class
    by simp
  with I show ⟨a,1⟩ ∈ r by (rule equiv_is_sym)
next assume ⟨a,1⟩ ∈ r
  moreover from A1 A3 have equiv(G,r)
    using IsAnormalSubgroup_def Group_ZF_2_4_L3
    by simp
  ultimately have r{a} = r{1}
    using equiv_class_eq by simp
  with groupAssum A1 A3 A4 show r{a} = e
    using Group_ZF_2_4_L5B by simp
qed
```

The class of $a \in G$ is the neutral element of the quotient G/H iff $a \in H$.

```
lemma (in group0) Group_ZF_2_4_L5E:
  assumes IsAnormalSubgroup(G,f,H) and
  a∈G and r = QuotientGroupRel(G,f,H) and
  TheNeutralElement(G//r,QuotientGroupOp(G,f,H)) = e
  shows r{a} = e ⟷ a∈H
  using prems Group_ZF_2_4_L5C  Group_ZF_2_4_L5D
  by simp
```

Essential condition to show that every subgroup of an abelian group is normal.

```
lemma (in group0) Group_ZF_2_4_L5:
  assumes A1:f {is commutative on} G
  and A2:IsAsubgroup(H,f)
  and A3:g∈G h∈H
  shows g·h·g⁻¹ ∈ H
proof -
  from A2 A3 have T1:h∈G g⁻¹ ∈ G
    using group0_3_L2 inverse_in_group by auto
  with A3 A1 have g·h·g⁻¹ = g⁻¹·g·h
    using group0_4_L4A by simp
  with A3 T1 show thesis using
    group0_2_L6 group0_2_L2
    by simp
qed
```

Every subgroup of an abelian group is normal. Moreover, the quotient group is also abelian.

```
lemma Group_ZF_2_4_L6:
```

```
  assumes A1: IsAgroup(G,f)
  and A2: f {is commutative on} G
  and A3: IsAsubgroup(H,f)
  shows  IsAnormalSubgroup(G,f,H)
  QuotientGroupOp(G,f,H) {is commutative on} (G//QuotientGroupRel(G,f,H))
proof -
  from A1 A2 A3 show T1: IsAnormalSubgroup(G,f,H) using
    group0_def IsAnormalSubgroup_def group0.Group_ZF_2_4_L5
    by simp
  let r = QuotientGroupRel(G,f,H)
  from A1 A3 T1 have equiv(G,r) Congruent2(r,f)
    using group0_def group0.Group_ZF_2_4_L3 Group_ZF_2_4_L5A
    by auto
  with A2 show
    QuotientGroupOp(G,f,H) {is commutative on} (G//QuotientGroupRel(G,f,H))
    using EquivClass_2_T1 QuotientGroupOp_def
    by simp
qed
```

The group inverse (in the quotient group) of a class (coset) is the class of the inverse.

```
lemma (in group0) Group_ZF_2_4_L7:
  assumes IsAnormalSubgroup(G,f,H)
  and a∈G and r = QuotientGroupRel(G,f,H)
  and F = QuotientGroupOp(G,f,H)
  shows r{a⁻¹} = GroupInv(G//r,F)(r{a})
  using groupAssum prems IsAnormalSubgroup_def Group_ZF_2_4_L3
    Group_ZF_2_4_L5A QuotientGroupOp_def Group_ZF_2_2_L4
  by simp
```

15.4 Function spaces as monoids

On every space of functions $\{f : X \to X\}$ we can define a natural monoid structure with composition as the operation. This section explores this fact.

The next lemma states that composition has a neutral element, namely the identity function on X (the one that maps $x \in X$ into itself).

```
lemma Group_ZF_2_5_L1: assumes A1: F = Composition(X)
  shows ∃I∈(X→X). ∀f∈(X→X). F<I,f> = f ∧ F<f,I> = f
proof-
  let I = id(X)
  from A1 have
    I ∈ X→X ∧ (∀f∈(X→X). F<I,f> = f ∧ F<f,I> = f)
    using id_type func_ZF_6_L1A by simp
  thus thesis by auto
qed
```

The space of functions that map a set X into itsef is a monoid with composition as operation and the identity function as the neutral element.

```
lemma Group_ZF_2_5_L2: shows
  IsAmonoid(X→X,Composition(X))
  id(X) = TheNeutralElement(X→X,Composition(X))
proof -
  let I = id(X)
  let F = Composition(X)
  show IsAmonoid(X→X,Composition(X))
    using func_ZF_5_L5 Group_ZF_2_5_L1 IsAmonoid_def
    by auto
  then have monoid0(X→X,F)
    using monoid0_def by simp
  moreover have
    I ∈ X→X ∧ (∀f∈(X→X). F<I,f> = f ∧ F<f,I> = f)
    using id_type func_ZF_6_L1A by simp
  ultimately show I = TheNeutralElement(X→X,F)
    using monoid0.group0_1_L4 by auto
qed
```

This concludes Group_ZF_2 theory.

end

16 Group_ZF_3.thy

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.

16.1 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_x.thy 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 = f {lifted to function space over} X
  and
  A2:s ∈ FinRangeFunctions(X,G) r ∈  FinRangeFunctions(X,G)
  shows F<s,r> ∈ FinRangeFunctions(X,G)
proof -
  let q = F<s,r>
  from A2 have T1:s:X→G r:X→G
    using FinRangeFunctions_def by auto
  with A1 have T2:q : X→G
    using group0_2_L1 monoid0.Group_ZF_2_1_L0
    by simp
  moreover have q(X) ∈ Fin(G)
  proof -
    from A2 have
      {s(x). x∈X} ∈ Fin(G)
      {r(x). x∈X} ∈ Fin(G)
      using Finite1_L18 by auto
    with A1 T1 T2 show thesis using
      group_oper_assocA Finite1_L15 Group_ZF_2_1_L3 func_imagedef
      by simp
  qed
  ultimately show thesis using FinRangeFunctions_def
    by simp
qed
```

The set of group valued finite range functions is closed with respect to the lifted group operation.

```
lemma (in group0) Group_ZF_3_1_L2:
  assumes A1: F = f {lifted to function space over} X
```

```
  shows FinRangeFunctions(X,G) {is closed under} F
proof -
  let A = FinRangeFunctions(X,G)
  from A1 have ∀x∈A. ∀y∈A. F<x,y> ∈ A
    using Group_ZF_3_1_L1 by simp
  then show thesis using IsOpClosed_def by simp
qed
```

A composition of a finite range function with the group inverse is a finite range function.

```
lemma (in group0) Group_ZF_3_1_L3:
  assumes A1: s ∈ FinRangeFunctions(X,G)
  shows GroupInv(G,f) O s ∈ FinRangeFunctions(X,G)
  using groupAssum prems group0_2_T2 Finite1_L20 by simp
```

The set of finite range functions is s subgroup of the lifted group.

```
theorem Group_ZF_3_1_T1:
  assumes A1:IsAgroup(G,f)
  and A2:F = f {lifted to function space over} X
  and A3:X≠0
  shows IsAsubgroup(FinRangeFunctions(X,G),F)
proof -
  let e = TheNeutralElement(G,f)
  let S = FinRangeFunctions(X,G)
  from A1 have T1:group0(G,f) using group0_def
    by simp
  with A1 A2 have T2:group0(X→G,F)
    using group0.Group_ZF_2_1_T2 group0_def
    by simp
  moreover have S ≠ 0
  proof -
    from T1 A3 have
      ConstantFunction(X,e) ∈ S
      using group0.group0_2_L1 monoid0.group0_1_L3
        Finite1_L17 by simp
    thus thesis by auto
  qed
  moreover have S ⊆ X→G
    using FinRangeFunctions_def by auto
  moreover from A2 T1 have
    S {is closed under} F
    using group0.Group_ZF_3_1_L2
    by simp
  moreover from A1 A2 T1 have
    ∀s ∈ S. GroupInv(X→G,F)(s) ∈ S
    using FinRangeFunctions_def group0.Group_ZF_2_1_L6
      group0.Group_ZF_3_1_L3 by simp
  ultimately show thesis
    using group0.group0_3_T3 by simp
```

qed

16.2 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. Almost homomorphisms are defined as those maps $s: G\rightarrow G$ such that the homomorphism difference takes only finite number of values on $G\times G$. Although almost homomorphisms can be in principle defined on a monoid with values in a group, we limit ourselves to the situation where the monoid and the group are the same. The set of slopes related to a specific group is called AlmostHoms(G,f). 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 $G\rightarrow G$ and restricted to the set 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_x.thy series.

constdefs
```
HomDiff(G,f,s,x) ≡
f⟨s(f<fst(x),snd(x)>) ,
(GroupInv(G,f)(f<s(fst(x)),s(snd(x))>))⟩

AlmostHoms(G,f) ≡
{s ∈ G→G.{HomDiff(G,f,s,x). x ∈ G×G } ∈ Fin(G)}

AlHomOp1(G,f) ≡
restrict(f {lifted to function space over} G,
AlmostHoms(G,f)×AlmostHoms(G,f))

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) Group_ZF_3_2_L1:
  shows HomDiff(G,f,s,<m,n>) = s(m·n)·(s(m)·s(n))⁻¹
  using HomDiff_def by simp
```

The next lemma shows the set from the definition of almost homomorphism in a different form.

```
lemma (in group0) Group_ZF_3_2_L1A:
  {HomDiff(G,f,s,x). x ∈ G×G } = {s(m·n)·(s(m)·s(n))⁻¹. <m,n> ∈ G×G}
proof -
  have ∀m∈G.∀n∈G. HomDiff(G,f,s,<m,n>) = s(m·n)·(s(m)·s(n))⁻¹
    using Group_ZF_3_2_L1 by simp
  then show thesis by (rule ZF1_1_L4A)
qed
```

Let's define some notation. We inherit the notation and assumptions from the group0 context (locale) and add some. We will use AH to denote the set of almost homomorphisms. $\sim$ is the inverse (negative if the group is the group of integers) of almost homomorphisms, $(\sim p)(n) = p(n)^{-1}$. δ will denote the homomorphism difference specific for the group (HomDiff(G, f)). The notation $s \approx r$ will mean that s, r are almost equal, that is they are in the equivalence relation defined by the group of finite range functions (that is a normal subgroup of almost homomorphisms, if the group is abelian). We show that this is equivalent to the set $\{s(n) \cdot r(n)^{-1} : n \in G\}$ being finite. We also add an assumption that the G is abelian as many needed properties do not hold without that.

```
locale group1 = group0 +
  assumes isAbelian: f {is commutative on} G

  fixes AH
  defines AH_def [simp]: AH ≡ AlmostHoms(G,f)

  fixes Op1
  defines Op1_def [simp]: Op1 ≡ AlHomOp1(G,f)

  fixes Op2
  defines Op2_def [simp]: Op2 ≡ AlHomOp2(G,f)

  fixes FR
  defines FR_def [simp]: FR ≡ FinRangeFunctions(G,G)

  fixes neg :: i⇒i (∼_ [90] 91)
  defines neg_def [simp]: ∼s ≡ GroupInv(G,f) O s

  fixes δ
  defines δ_def [simp]: δ(s,x) ≡ HomDiff(G,f,s,x)
```

```
  fixes AHprod (infix · 69)
  defines AHprod_def [simp]: s · r ≡  AlHomOp1(G,f)<s,r>

  fixes AHcomp (infix ∘ 70)
  defines AHcomp_def [simp]: s ∘ r ≡  AlHomOp2(G,f)<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 = f {lifted to function space over} G
  shows δ(F<s,r>,x) = δ(s,x)·δ(r,x)
proof -
  let p = F<s,r>
  from A2 obtain m n where
    D1: x = <m,n> m∈G n∈G
    by auto
  then have T1:m·n ∈ G
    using group0_2_L1 monoid0.group0_1_L1 by simp
  with A1 D1 have T2:
    s(m)∈G s(n)∈G r(m)∈G
    r(n)∈G s(m·n)∈G r(m·n)∈G
    using apply_funtype by auto
  from A3 A1 have T3:p : G→G
    using group0_2_L1 monoid0.Group_ZF_2_1_L0
    by simp
  from D1 T3 have
    δ(p,x) = p(m·n)·((p(n))^-1·(p(m))^-1)
    using Group_ZF_3_2_L1 apply_funtype group_inv_of_two
    by simp
  also from A3 A1 D1 T1 isAbelian T2 have
    ... = δ(s,x)· δ(r,x)
    using Group_ZF_2_1_L3 group0_4_L3  Group_ZF_3_2_L1
    by simp
  finally show thesis by simp
qed
```

The group operation lifted to the function space over G preserves almost homomorphisms.

```
lemma (in group1) Group_ZF_3_2_L2: assumes A1: s ∈ AH r ∈ AH
  and A2: F = f {lifted to function space over} G
  shows F<s,r> ∈ AH
proof -
  let p = F<s,r>
  from A1 A2 have p : G→G
```

```
    using AlmostHoms_def group0_2_L1 monoid0.Group_ZF_2_1_L0
    by simp
  moreover have
    {δ(p,x). x ∈ G×G} ∈ Fin(G)
  proof -
    from A1 have
      {δ(s,x). x ∈ G×G } ∈ Fin(G)
      {δ(r,x). x ∈ G×G } ∈ Fin(G)
      using AlmostHoms_def by auto
    with groupAssum A1 A2 show thesis
      using IsAgroup_def IsAmonoid_def IsAssociative_def
      Finite1_L15 AlmostHoms_def Group_ZF_3_2_L1
      by auto
  qed
  ultimately show thesis using AlmostHoms_def
    by simp
qed
```

The set of almost homomorphisms is closed under the lifted group operation.

```
lemma (in group1) Group_ZF_3_2_L3:
  assumes F = f {lifted to function space over} G
  shows AH {is closed under} F
  using prems IsOpClosed_def Group_ZF_3_2_L2 by simp
```

The terms in the homomorphism difference for a function are in the group.

```
lemma (in group1) Group_ZF_3_2_L4:
  assumes s:G→G and m∈G  n∈G
  shows
  m·n ∈ G
  s(m·n) ∈ G
  s(m) ∈ G s(n) ∈ G
  δ(s,<m,n>) ∈ G
  s(m)·s(n) ∈ G
  using prems group_op_closed inverse_in_group
    apply_funtype HomDiff_def by auto
```

It is handy to have a version of `Group_ZF_3_2_L4` specifically for almost homomorphisms.

```
corollary (in group1) Group_ZF_3_2_L4A:
  assumes s ∈ AH and m∈G  n∈G
  shows m·n ∈ G
  s(m·n) ∈ G
  s(m) ∈ G s(n) ∈ G
  δ(s,<m,n>) ∈ G
  s(m)·s(n) ∈ G
  using prems AlmostHoms_def Group_ZF_3_2_L4
  by auto
```

The terms in the homomorphism difference are in the group, a different form.

```
lemma (in group1) Group_ZF_3_2_L4B:
  assumes A1:s ∈ AH and A2:x∈G×G
  shows fst(x)·snd(x) ∈ G
  s(fst(x)·snd(x)) ∈ G
  s(fst(x)) ∈ G s(snd(x)) ∈ G
  δ(s,x) ∈ G
  s(fst(x))·s(snd(x)) ∈ G
proof -
  let m = fst(x)
  let n = snd(x)
  from A1 A2 show
    m·n ∈ G  s(m·n) ∈ G
    s(m) ∈ G s(n) ∈ G
    s(m)·s(n) ∈ G
    using Group_ZF_3_2_L4A
    by auto
  from A1 A2 have δ(s,<m,n>) ∈ G using Group_ZF_3_2_L4A
    by simp
  moreover from A2 have <m,n> = x by auto
  ultimately show δ(s,x) ∈ G by simp
qed
```

What are the values of the inverse of an almost homomorphism?

```
lemma (in group1) Group_ZF_3_2_L5:
  assumes s ∈ AH and n∈G
  shows (∼s)(n) = (s(n))⁻¹
  using prems AlmostHoms_def comp_fun_apply by auto
```

Homomorphism difference commutes with the inverse for almost homomorphisms.

```
lemma (in group1) Group_ZF_3_2_L6:
  assumes A1:s ∈ AH and A2:x∈G×G
  shows δ(∼s,x) = (δ(s,x))⁻¹
proof -
  let m = fst(x)
  let n = snd(x)
  have δ(∼s,x) = (∼s)(m·n)·((∼s)(m)·(∼s)(n))⁻¹
    using HomDiff_def by simp
  from A1 A2 isAbelian show thesis
    using Group_ZF_3_2_L4B HomDiff_def
      Group_ZF_3_2_L5 group0_4_L4A
    by simp
qed
```

The inverse of an almost homomorphism maps the group into itself.

```
lemma (in group1) Group_ZF_3_2_L7:
  assumes s ∈ AH
```

```
  shows ∼s : G→G
  using groupAssum prems AlmostHoms_def group0_2_T2 comp_fun by auto
```

The inverse of an almost homomorphism is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_2_L8:
  assumes A1: F = f {lifted to function space over} G
  and A2: s ∈ AH
  shows GroupInv(G→G,F)(s) ∈ AH
proof -
  from A2 have {δ(s,x). x ∈ G×G} ∈ Fin(G)
    using AlmostHoms_def by simp
  with groupAssum  have
    GroupInv(G,f){δ(s,x). x ∈ G×G} ∈ Fin(G)
    using group0_2_T2 Finite1_L6A by blast
  moreover have
     GroupInv(G,f){δ(s,x). x ∈ G×G} =
    {(δ(s,x))⁻¹. x ∈ G×G}
  proof -
    from groupAssum have
      GroupInv(G,f) : G→G
      using group0_2_T2 by simp
    moreover from A2 have
      ∀x∈G×G. δ(s,x)∈G
      using Group_ZF_3_2_L4B by simp
    ultimately show thesis
      using func1_1_L17 by simp
  qed
  ultimately have {(δ(s,x))⁻¹. x ∈ G×G} ∈ Fin(G)
    by simp
  moreover from A2 have
    {(δ(s,x))⁻¹. x ∈ G×G} = {δ(∼s,x). x ∈ G×G}
    using Group_ZF_3_2_L6 by simp
  ultimately have {δ(∼s,x). x ∈ G×G} ∈ Fin(G)
    by simp
  with A2 groupAssum A1 show thesis
    using Group_ZF_3_2_L7 AlmostHoms_def Group_ZF_2_1_L6
    by simp
qed
```

The function that assigns the neutral element everywhere is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_2_L9:
  ConstantFunction(G,1) ∈ AH
  AH≠0
proof -
  let z = ConstantFunction(G,1)
  have G×G≠0 using group0_2_L1 monoid0.group0_1_L3A
    by blast
  moreover have ∀x∈G×G. δ(z,x) = 1
```

```
  proof
    fix x assume A1:x ∈ G × G
    then obtain m n where x = <m,n> m∈G n∈G
      by auto
    then show δ(z,x) = 1
      using group0_2_L1 monoid0.group0_1_L1
        func1_3_L2 HomDiff_def group0_2_L2
        group_inv_of_one by simp
  qed
  ultimately have {δ(z,x). x∈G×G} = {1} by (rule ZF1_1_L5)
  then show z ∈ AH using group0_2_L2 Finite1_L16
    func1_3_L1 group0_2_L2 AlmostHoms_def by simp
  then show AH≠0 by auto
qed
```

If the group is abelian, then almost homomorphisms form a subgroup of the lifted group.

```
lemma Group_ZF_3_2_L10:
  assumes A1: IsAgroup(G,f)
  and A2: f {is commutative on} G
  and A3: F = f {lifted to function space over} G
  shows IsAsubgroup(AlmostHoms(G,f),F)
proof -
  let AH = AlmostHoms(G,f)
  from A2 A1 have T1:group1(G,f)
    using group1_axioms.intro group0_def group1_def
    by simp
  from A1 A3 have group0(G→G,F)
    using group0_def group0.Group_ZF_2_1_T2 by simp
  moreover from T1 have AH≠0
    using group1.Group_ZF_3_2_L9 by simp
  moreover have T2:AH ⊆ G→G
    using AlmostHoms_def by auto
  moreover from T1 A3 have
    AH {is closed under} F
    using group1.Group_ZF_3_2_L3 by simp
  moreover from T1 A3 have
    ∀s∈AH. GroupInv(G→G,F)(s) ∈ AH
    using group1.Group_ZF_3_2_L8 by simp
  ultimately show IsAsubgroup(AlmostHoms(G,f),F)
    using group0.group0_3_T3 by simp
qed
```

If the group is abelian, then almost homomorphisms form a group with the first operation, hence we can use theorems proven in group0 context aplied to this group.

```
lemma (in group1) Group_ZF_3_2_L10A:
  shows IsAgroup(AH,Op1) group0(AH,Op1)
    using groupAssum isAbelian Group_ZF_3_2_L10 IsAsubgroup_def
```

```
      AlHomOp1_def group0_def by auto
```

The group of almost homomorphisms is abelian

```
lemma Group_ZF_3_2_L11: assumes A1: IsAgroup(G,f)
  and A2: f {is commutative on} G
  shows
  IsAgroup(AlmostHoms(G,f),AlHomOp1(G,f))
  AlHomOp1(G,f) {is commutative on} AlmostHoms(G,f)
proof-
  let AH = AlmostHoms(G,f)
  let F = f {lifted to function space over} G
  from A1 A2 have IsAsubgroup(AH,F)
    using Group_ZF_3_2_L10 by simp
  then show IsAgroup(AH,AlHomOp1(G,f))
    using IsAsubgroup_def AlHomOp1_def by simp
  from A1 have F : (G→G)×(G→G)→(G→G)
    using IsAgroup_def monoid0_def monoid0.Group_ZF_2_1_L0A
    by simp
  moreover have AH ⊆ G→G
    using AlmostHoms_def by auto
  moreover from A1 A2 have
    F {is commutative on} (G→G)
    using group0_def group0.Group_ZF_2_1_L7
    by simp
  ultimately show
    AlHomOp1(G,f){is commutative on} AH
    using func_ZF_4_L1 AlHomOp1_def by simp
qed
```

The first operation on homomorphisms acts in a natural way on its operands.

```
lemma (in group1) Group_ZF_3_2_L12:
  assumes s∈AH  r∈AH and n∈G
  shows (s·r)(n) = s(n)·r(n)
  using prems AlHomOp1_def restrict AlmostHoms_def Group_ZF_2_1_L3
  by simp
```

What is the group inverse in the group of almost homomorphisms?

```
lemma (in group1) Group_ZF_3_2_L13:
  assumes A1: s∈AH
  shows
  GroupInv(AH,Op1)(s) = GroupInv(G,f) O s
  GroupInv(AH,Op1)(s) ∈ AH
  GroupInv(G,f) O s ∈ AH
proof -
  let F = f {lifted to function space over} G
  from groupAssum isAbelian have IsAsubgroup(AH,F)
    using Group_ZF_3_2_L10 by simp
  with A1 show I: GroupInv(AH,Op1)(s) = GroupInv(G,f) O s
    using AlHomOp1_def Group_ZF_2_1_L6A by simp
```

```
  from A1 show GroupInv(AH,Op1)(s) ∈ AH
    using Group_ZF_3_2_L10A group0.inverse_in_group by simp
  with I show GroupInv(G,f) O s ∈ AH by simp
qed
```

The group inverse in the group of almost homomorphisms acts in a natural way on its operand.

```
lemma (in group1) Group_ZF_3_2_L14:
  assumes s∈AH and n∈G
  shows (GroupInv(AH,Op1)(s))(n) = (s(n))⁻¹
  using isAbelian prems Group_ZF_3_2_L13 AlmostHoms_def comp_fun_apply
  by auto
```

The next lemma states that if s, r are almost homomorphisms, then $s \cdot r^{-1}$ is also an almost homomorphism.

```
lemma Group_ZF_3_2_L15: assumes IsAgroup(G,f)
  and f {is commutative on} G
  and AH = AlmostHoms(G,f) Op1 = AlHomOp1(G,f)
  and s ∈ AH  r ∈ AH
  shows
  Op1<s,r> ∈ AH
  GroupInv(AH,Op1)(r) ∈ AH
  Op1<s,GroupInv(AH,Op1)(r)> ∈ AH
  using prems group0_def group1_axioms.intro group1_def
      group1.Group_ZF_3_2_L10A group0.group0_2_L1
      monoid0.group0_1_L1 group0.inverse_in_group by auto
```

A version of Group_ZF_3_2_L15 formulated in notation used in group1 context. States that the product of almost homomorphisms is an almost homomorphism and the the product of an almost homomorphism with a (pointwise) inverse of an almost homomorphism is an almost homomorphism.

```
corollary (in group1) Group_ZF_3_2_L16: assumes s ∈ AH  r ∈ AH
  shows s·r ∈ AH   s·(∼r) ∈ AH
  using prems isAbelian group0_def group1_axioms.intro group1_def
  Group_ZF_3_2_L15  Group_ZF_3_2_L13 by auto
```

16.3 The classes of almost homomorphisms

In the Real_ZF_x 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: FR ⊆ AH
proof
  fix s assume A1:s ∈ FR
  then have T1:{s(n). n ∈ G} ∈ Fin(G)
```

```
    {s(fst(x)). x∈G×G} ∈ Fin(G)
    {s(snd(x)). x∈G×G} ∈ Fin(G)
    using Finite1_L18 Finite1_L6B by auto
  have {s(fst(x)·snd(x)). x ∈ G×G} ∈ Fin(G)
  proof -
    have ∀x∈G×G. fst(x)·snd(x) ∈ G
      using group0_2_L1 monoid0.group0_1_L1 by simp
    moreover from T1 have {s(n). n ∈ G} ∈ Fin(G) by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  moreover have
    {(s(fst(x))·s(snd(x)))⁻¹. x∈G×G} ∈ Fin(G)
  proof -
    have ∀g∈G. g⁻¹ ∈ G using inverse_in_group
      by simp
    moreover from T1 have
      {s(fst(x))·s(snd(x)). x∈G×G} ∈ Fin(G)
      using group_oper_assocA  Finite1_L15 by simp
    ultimately show thesis
      by (rule Finite1_L6C)
  qed
  ultimately have {δ(s,x). x∈G×G} ∈ Fin(G)
    using HomDiff_def Finite1_L15  group_oper_assocA
    by simp
  with A1 show s ∈ AH
    using FinRangeFunctions_def AlmostHoms_def
    by simp
qed
```

Finite range functions valued in an abelian group form a normal subgroup of almost homomorphisms.

```
lemma Group_ZF_3_3_L2: assumes A1:IsAgroup(G,f)
  and A2:f {is commutative on} G
  shows
  IsAsubgroup(FinRangeFunctions(G,G),AlHomOp1(G,f))
  IsAnormalSubgroup(AlmostHoms(G,f),AlHomOp1(G,f),
  FinRangeFunctions(G,G))
proof -
  let H1 = AlmostHoms(G,f)
  let H2 = FinRangeFunctions(G,G)
  let F = f {lifted to function space over} G
  from A1 A2 have T1:group0(G,f)
    monoid0(G,f) group1(G,f)
    using group0_def group0.group0_2_L1
      group1_axioms.intro group1_def
    by auto
  with A1 A2 have IsAgroup(G→G,F)
    IsAsubgroup(H1,F) IsAsubgroup(H2,F)
    using group0.Group_ZF_2_1_T2 Group_ZF_3_2_L10
```

```
      monoid0.group0_1_L3A Group_ZF_3_1_T1
    by auto
  then have
    IsAsubgroup(H1∩H2,restrict(F,H1×H1))
    using group0_3_L7 by simp
  moreover from T1 have H1∩H2 = H2
    using group1.Group_ZF_3_3_L1 by auto
  ultimately show IsAsubgroup(H2,AlHomOp1(G,f))
    using AlHomOp1_def by simp
  with A1 A2 show IsAnormalSubgroup(AlmostHoms(G,f),AlHomOp1(G,f),
    FinRangeFunctions(G,G))
    using Group_ZF_3_2_L11 Group_ZF_2_4_L6
    by simp
qed
```

The group of almost homomorphisms divided by the subgroup of finite range functions is an abelian group.

```
theorem (in group1) Group_ZF_3_3_T1:
  shows
  IsAgroup(AH//QuotientGroupRel(AH,Op1,FR),QuotientGroupOp(AH,Op1,FR))
  and
  QuotientGroupOp(AH,Op1,FR) {is commutative on}
  (AH//QuotientGroupRel(AH,Op1,FR))
  using groupAssum isAbelian Group_ZF_3_3_L2 Group_ZF_3_2_L10A
    Group_ZF_2_4_T1 Group_ZF_3_2_L10A Group_ZF_3_2_L11
    Group_ZF_3_3_L2 IsAnormalSubgroup_def Group_ZF_2_4_L6 by auto
```

It is useful to have a direct statement that the quotient group relation is an equivalence relation for the group of AH and subgroup FR.

```
lemma (in group1) Group_ZF_3_3_L3:
  QuotientGroupRel(AH,Op1,FR) ⊆ AH × AH
  equiv(AH,QuotientGroupRel(AH,Op1,FR))
  using groupAssum isAbelian QuotientGroupRel_def
    Group_ZF_3_3_L2 Group_ZF_3_2_L10A group0.Group_ZF_2_4_L3
  by auto
```

The "almost equal" relation is symmetric.

```
lemma (in group1) Group_ZF_3_3_L3A: assumes A1: s≈r
  shows r≈s
proof -
  let R = QuotientGroupRel(AH,Op1,FR)
  from A1 have equiv(AH,R) and ⟨s,r⟩ ∈ R
    using Group_ZF_3_3_L3 by auto
  then have ⟨r,s⟩ ∈ R by (rule equiv_is_sym)
  then show r≈s by simp
qed
```

Although we have bypassed this fact when proving that group of almost homomorphisms divided by the subgroup of finite range functions is a group, it is still useful to know directly that the first group operation on AH is congruent with respect to the quotient group relation.

```
lemma (in group1) Group_ZF_3_3_L4:
  shows Congruent2(QuotientGroupRel(AH,Op1,FR),Op1)
  using groupAssum isAbelian Group_ZF_3_2_L10A Group_ZF_3_3_L2
    Group_ZF_2_4_L5A by simp
```

The class of an almost homomorphism s is the neutral element of the quotient group of almost homomorphisms iff s is a finite range function.

```
lemma (in group1) Group_ZF_3_3_L5: assumes s ∈ AH and
  r = QuotientGroupRel(AH,Op1,FR) and
  TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR)) = e
  shows r{s} = e ⟷ s ∈ FR
  using groupAssum isAbelian prems Group_ZF_3_2_L11
    group0_def Group_ZF_3_3_L2 group0.Group_ZF_2_4_L5E
  by simp
```

The group inverse of a class of an almost homomorphism f is the class of the inverse of f.

```
lemma (in group1) Group_ZF_3_3_L6:
  assumes A1: s ∈ AH  and
  r = QuotientGroupRel(AH,Op1,FR) and
  F = ProjFun2(AH,r,Op1)
  shows r{∼s} = GroupInv(AH//r,F)(r{s})
proof -
  from groupAssum isAbelian prems have
    r{GroupInv(AH, Op1)(s)} = GroupInv(AH//r,F)(r  {s})
    using Group_ZF_3_2_L10A Group_ZF_3_3_L2 QuotientGroupOp_def
      group0.Group_ZF_2_4_L7 by simp
  with A1 show thesis using Group_ZF_3_2_L13
    by simp
qed
```

16.4 Compositions of almost homomorphisms

The goal of this section is to establish some facts about composition of almost homomorphisms. needed for the real numbers construction in Real_ZF_x.thy serias. In particular we show that the set of almost homomorphisms is closed under composition and that composition is congruent with respect to the equivalence relation defined by the group of finite range functions (a normal subgroup of almost homomorphisms).

The next formula restates the definition of the homomorphism difference to express the value an almost homomorphism on a product.

```
lemma (in group1) Group_ZF_3_4_L1:
  assumes s∈AH and  m∈G  n∈G
  shows s(m·n) = s(m)·s(n)·δ(s,<m,n>)
```

```
  using isAbelian prems Group_ZF_3_2_L4A HomDiff_def group0_4_L5
  by simp
```

What is the value of a composition of almost homomorhisms?

```
lemma (in group1) Group_ZF_3_4_L2:
  assumes s∈AH  r∈AH and m∈G
  shows (s∘r)(m) = s(r(m))  s(r(m)) ∈ G
  using prems AlmostHoms_def func_ZF_5_L3 restrict AlHomOp2_def
    apply_funtype by auto
```

What is the homomorphism difference of a composition?

```
lemma (in group1) Group_ZF_3_4_L3:
  assumes A1: s∈AH  r∈AH and A2: m∈G  n∈G
  shows δ(s∘r,<m,n>) =
  δ(s,<r(m),r(n)>)·s(δ(r,<m,n>))·δ(s,<r(m)·r(n),δ(r,<m,n>)>)
proof -
  from A1 A2 have T1:
    s(r(m))· s(r(n)) ∈ G
    δ(s,<r(m),r(n)>)∈ G s(δ(r,<m,n>)) ∈G
    δ(s,<(r(m)·r(n)),δ(r,<m,n>)>) ∈ G
    using Group_ZF_3_4_L2 AlmostHoms_def apply_funtype
      Group_ZF_3_2_L4A group0_2_L1 monoid0.group0_1_L1
    by auto
  from A1 A2 have δ(s∘r,<m,n>) =
    s(r(m)·r(n)·δ(r,<m,n>))·(s((r(m)))·s(r(n)))⁻¹
    using HomDiff_def group0_2_L1 monoid0.group0_1_L1 Group_ZF_3_4_L2
      Group_ZF_3_4_L1 by simp
  moreover from A1 A2 have
    s(r(m)·r(n)·δ(r,<m,n>)) =
    s(r(m)·r(n))·s(δ(r,<m,n>))·δ(s,<(r(m)·r(n)),δ(r,<m,n>)>)
    s(r(m)·r(n)) = s(r(m))·s(r(n))·δ(s,<r(m),r(n)>)
    using Group_ZF_3_2_L4A Group_ZF_3_4_L1 by auto
  moreover from T1 isAbelian have
    s(r(m))·s(r(n))·δ(s,<r(m),r(n)>)·
    s(δ(r,<m,n>))·δ(s,<(r(m)·r(n)),δ(r,<m,n>)>)·
    (s((r(m)))·s(r(n)))⁻¹ =
    δ(s,<r(m),r(n)>)·s(δ(r,<m,n>))·δ(s,<(r(m)·r(n)),δ(r,<m,n>)>)
    using group0_4_L6C by simp
  ultimately show thesis by simp
qed
```

What is the homomorphism difference of a composition (another form)? Here we split the homomorphism difference of a composition into a product of three factors. This will help us in proving that the range of homomorphism difference for the composition is finite, as each factor has finite range.

```
lemma (in group1) Group_ZF_3_4_L4:
  assumes A1: s∈AH  r∈AH and A2: x ∈ G×G
  and A3:
```

```
  A = δ(s,<r(fst(x)),r(snd(x))>)
  B = s(δ(r,x))
  C = δ(s,<(r(fst(x))·r(snd(x))),δ(r,x)>)
  shows δ(s∘r,x) = A·B·C
proof -
  let m = fst(x)
  let n = snd(x)
  from A1 have s∈AH r∈AH .
  moreover from A2 have m∈G n∈G
    by auto
  ultimately have
    δ(s∘r,<m,n>) =
    δ(s,<r(m),r(n)>)·s(δ(r,<m,n>))·
    δ(s,<(r(m)·r(n)),δ(r,<m,n>)>)
    by (rule Group_ZF_3_4_L3)
  with A1 A2 A3 show thesis
    by auto
qed
```

The range of the homomorphism difference of a composition of two almost homomorphisms is finite. This is the essential condition to show that a composition of almost homomorphisms is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_4_L5:
  assumes A1: s∈AH  r∈AH
  shows {δ(Composition(G)<s,r>,x). x ∈ G×G} ∈ Fin(G)
proof -
  from A1 have
    ∀x∈G×G. <r(fst(x)),r(snd(x))> ∈ G×G
    using Group_ZF_3_2_L4B by simp
  moreover from A1 have
    {δ(s,x). x∈G×G} ∈ Fin(G)
    using AlmostHoms_def by simp
  ultimately have
    {δ(s,<r(fst(x)),r(snd(x))>). x∈G×G} ∈ Fin(G)
    by (rule Finite1_L6B)
  moreover have {s(δ(r,x)). x∈G×G} ∈ Fin(G)
  proof -
    from A1 have ∀m∈G. s(m) ∈ G
      using AlmostHoms_def apply_funtype by auto
    moreover from A1 have {δ(r,x). x∈G×G} ∈ Fin(G)
      using AlmostHoms_def by simp
    ultimately show thesis
      by (rule Finite1_L6C)
  qed
  ultimately have
    {δ(s,<r(fst(x)),r(snd(x))>)·s(δ(r,x)). x∈G×G} ∈ Fin(G)
    using group_oper_assocA Finite1_L15 by simp
  moreover have
    {δ(s,<(r(fst(x))·r(snd(x))),δ(r,x)>).  x∈G×G} ∈ Fin(G)
```

```
  proof -
    from A1 have
    ∀x∈G×G. <(r(fst(x))·r(snd(x))),δ(r,x)> ∈ G×G
      using Group_ZF_3_2_L4B by simp
    moreover from A1 have
      {δ(s,x). x∈G×G} ∈ Fin(G)
      using AlmostHoms_def by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  ultimately have
    {δ(s,<r(fst(x)),r(snd(x))>)·s(δ(r,x))·
    δ(s,<(r(fst(x))·r(snd(x))),δ(r,x)>). x∈G×G} ∈ Fin(G)
    using group_oper_assocA Finite1_L15 by simp
  moreover from A1 have {δ(s∘r,x). x∈G×G} =
    {δ(s,<r(fst(x)),r(snd(x))>)·s(δ(r,x))·
    δ(s,<(r(fst(x))·r(snd(x))),δ(r,x)>). x∈G×G}
    using Group_ZF_3_4_L4 by simp
  ultimately have {δ(s∘r,x). x∈G×G} ∈ Fin(G) by simp
  with A1 show thesis using restrict AlHomOp2_def
    by simp
qed
```

Composition of almost homomorphisms is an almost homomorphism.

```
theorem (in group1) Group_ZF_3_4_T1:
  assumes A1: s∈AH  r∈AH
  shows Composition(G)<s,r> ∈ AH s∘r ∈ AH
proof -
  from A1 have <s,r> ∈ (G→G)×(G→G)
    using AlmostHoms_def by simp
  then have Composition(G)<s,r> : G→G
    using func_ZF_5_L1 apply_funtype by blast
  with A1 show Composition(G)<s,r> ∈ AH
    using Group_ZF_3_4_L5 AlmostHoms_def
    by simp
  with A1 show  s∘r ∈ AH using AlHomOp2_def restrict
    by simp
qed
```

The set of almost homomorphisms is closed under composition. The second operation on almost homomorphisms is associative.

```
lemma (in group1) Group_ZF_3_4_L6: shows
  AH {is closed under} Composition(G)
  AlHomOp2(G,f) {is associative on} AH
proof -
  show AH {is closed under} Composition(G)
    using Group_ZF_3_4_T1 IsOpClosed_def by simp
  moreover have AH ⊆ G→G using AlmostHoms_def
    by auto
  moreover have
```

```
    Composition(G) {is associative on} (G→G)
    using func_ZF_5_L5 by simp
  ultimately show AlHomOp2(G,f) {is associative on} AH
    using func_ZF_4_L3 AlHomOp2_def by simp
qed
```

Type information related to the situation of two almost homomorphisms.

```
lemma (in group1) Group_ZF_3_4_L7:
  assumes A1: s∈AH  r∈AH and A2: n∈G
  shows
  s(n) ∈ G (r(n))⁻¹ ∈ G
  s(n)·(r(n))⁻¹ ∈ G   s(r(n)) ∈ G
proof -
  from A1 A2 show
    s(n) ∈ G
    (r(n))⁻¹ ∈ G
    s(r(n)) ∈ G
    s(n)·(r(n))⁻¹ ∈ G
    using AlmostHoms_def apply_type
      group0_2_L1 monoid0.group0_1_L1 inverse_in_group
    by auto
qed
```

Type information related to the situation of three almost homomorphisms.

```
lemma (in group1) Group_ZF_3_4_L8:
  assumes A1: s∈AH  r∈AH  q∈AH and A2: n∈G
  shows
  q(n)∈G
  s(r(n)) ∈ G
  r(n)·(q(n))⁻¹ ∈ G
  s(r(n)·(q(n))⁻¹) ∈ G
  δ(s,<q(n),r(n)·(q(n))⁻¹>) ∈ G
proof -
  from A1 A2 show
    q(n)∈ G  s(r(n)) ∈ G r(n)·(q(n))⁻¹ ∈ G
    using AlmostHoms_def apply_type
      group0_2_L1 monoid0.group0_1_L1 inverse_in_group
    by auto
  with A1 A2 show s(r(n)·(q(n))⁻¹) ∈ G
    δ(s,<q(n),r(n)·(q(n))⁻¹>) ∈ G
    using AlmostHoms_def apply_type Group_ZF_3_2_L4A
    by auto
qed
```

A formula useful in showing that the composition of almost homomorphisms is congruent with respect to the quotient group relation.

```
lemma (in group1) Group_ZF_3_4_L9:
  assumes A1: s1 ∈ AH  r1 ∈ AH  s2 ∈ AH  r2 ∈ AH
  and A2: n∈G
```

```
  shows (s1∘r1)(n)·((s2∘r2)(n))⁻¹ =
  s1(r2(n))· (s2(r2(n)))⁻¹·s1(r1(n)·(r2(n))⁻¹)·
  δ(s1,<r2(n),r1(n)·(r2(n))⁻¹>)
proof -
  from A1 A2 isAbelian have
    (s1∘r1)(n)·((s2∘r2)(n))⁻¹ =
    s1(r2(n)·(r1(n)·(r2(n))⁻¹))·(s2(r2(n)))⁻¹
    using Group_ZF_3_4_L2 Group_ZF_3_4_L7 group0_4_L6A
      group_oper_assoc by simp
  with A1 A2 have (s1∘r1)(n)·((s2∘r2)(n))⁻¹ = s1(r2(n))·
    s1(r1(n)·(r2(n))⁻¹)·δ(s1,<r2(n),r1(n)·(r2(n))⁻¹>)·
    (s2(r2(n)))⁻¹
    using Group_ZF_3_4_L8 Group_ZF_3_4_L1 by simp
  with A1 A2 isAbelian show thesis using
    Group_ZF_3_4_L8 group0_4_L7 by simp
qed
```

The next lemma shows a formula that translates an expression in terms of the first group operation on almost homomorphisms and the group inverse in the group of almost homomorphisms to an expression using only the underlying group operations.

```
lemma (in group1) Group_ZF_3_4_L10: assumes A1: s ∈ AH  r ∈ AH
  and A2: n ∈ G
  shows (s•(GroupInv(AH,Op1)(r)))(n) = s(n)·(r(n))⁻¹
proof -
  from isAbelian A1 A2 show thesis
    using Group_ZF_3_2_L13 Group_ZF_3_2_L12 Group_ZF_3_2_L14
    by simp
qed
```

A neccessary condition for two a. h. to be almost equal.

```
lemma (in group1) Group_ZF_3_4_L11:
  assumes A1: s≈r
  shows {s(n)·(r(n))⁻¹. n∈G} ∈ Fin(G)
proof -
  from A1 have s∈AH r∈AH
    using QuotientGroupRel_def by auto
  moreover from A1 have
    {(s•(GroupInv(AH,Op1)(r)))(n). n∈G} ∈ Fin(G)
    using QuotientGroupRel_def Finite1_L18 by simp
  ultimately show thesis
    using Group_ZF_3_4_L10 by simp
qed
```

A sufficient condition for two a. h. to be almost equal.

```
lemma (in group1) Group_ZF_3_4_L12: assumes A1: s∈AH  r∈AH
  and A2: {s(n)·(r(n))⁻¹. n∈G} ∈ Fin(G)
  shows s≈r
```

```
proof -
  from groupAssum isAbelian A1 A2 show thesis
    using Group_ZF_3_2_L15 AlmostHoms_def
    Group_ZF_3_4_L10 Finite1_L19 QuotientGroupRel_def
    by simp
qed
```

Another sufficient consdition for two a.h. to be almost equal. It is actually just an expansion of the definition of the quotient group relation.

```
lemma (in group1) Group_ZF_3_4_L12A: assumes s∈AH  r∈AH
  and s·(GroupInv(AH,Op1)(r)) ∈ FR
  shows s≈r  r≈s
proof  -
  from prems show s≈r using prems QuotientGroupRel_def
    by simp
  then show r≈s by (rule Group_ZF_3_3_L3A)
qed
```

Another necessary condition for two a.h. to be almost equal. It is actually just an expansion of the definition of the quotient group relation.

```
lemma (in group1) Group_ZF_3_4_L12B: assumes s≈r
  shows s·(GroupInv(AH,Op1)(r)) ∈ FR
  using prems QuotientGroupRel_def by simp
```

The next lemma states the essential condition for the composition of a. h. to be congruent with respect to the quotient group relation for the subgroup of finite range functions.

```
lemma (in group1) Group_ZF_3_4_L13:
  assumes A1: s1≈s2  r1≈r2
  shows (s1∘r1) ≈ (s2∘r2)
proof -
  have {s1(r2(n))· (s2(r2(n)))⁻¹. n∈G} ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. r2(n) ∈ G
      using QuotientGroupRel_def AlmostHoms_def apply_funtype
      by auto
    moreover from A1 have {s1(n)·(s2(n))⁻¹. n∈G} ∈ Fin(G)
      using Group_ZF_3_4_L11 by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  moreover have {s1(r1(n)·(r2(n))⁻¹). n ∈ G} ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. s1(n)∈G
      using QuotientGroupRel_def AlmostHoms_def apply_funtype
      by auto
    moreover from A1 have {r1(n)·(r2(n))⁻¹. n∈G} ∈ Fin(G)
      using Group_ZF_3_4_L11 by simp
    ultimately show thesis by (rule Finite1_L6C)
```

```
  qed
  ultimately have
    {s1(r2(n))· (s2(r2(n)))⁻¹·s1(r1(n)·(r2(n))⁻¹).
    n∈G} ∈ Fin(G)
    using group_oper_assocA Finite1_L15 by simp
  moreover have
    {δ(s1,<r2(n),r1(n)·(r2(n))⁻¹>). n∈G} ∈ Fin(G)
  proof -
    from A1 have ∀n∈G. <r2(n),r1(n)·(r2(n))⁻¹> ∈ G×G
      using QuotientGroupRel_def Group_ZF_3_4_L7 by auto
    moreover from A1 have {δ(s1,x). x ∈ G×G} ∈ Fin(G)
      using QuotientGroupRel_def AlmostHoms_def by simp
    ultimately show thesis by (rule Finite1_L6B)
  qed
  ultimately have
    {s1(r2(n))· (s2(r2(n)))⁻¹·s1(r1(n)·(r2(n))⁻¹)·
    δ(s1,<r2(n),r1(n)·(r2(n))⁻¹>). n∈G} ∈ Fin(G)
    using group_oper_assocA Finite1_L15 by simp
  with A1 show thesis using
    QuotientGroupRel_def Group_ZF_3_4_L9
    Group_ZF_3_4_T1 Group_ZF_3_4_L12 by simp
qed
```

Composition of a. h. to is congruent with respect to the quotient group relation for the subgroup of finite range functions. Recall that if an operation say "$\circ$" 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:
  Congruent2(QuotientGroupRel(AH,Op1,FR),Op2)
proof -
  show thesis using Group_ZF_3_4_L13 Congruent2_def
    by simp
qed
```

The homomorphism difference for the identity function is equal to the neutral element of the group (denoted e in the group1 context).

```
lemma (in group1) Group_ZF_3_4_L14: assumes A1: x ∈ G×G
  shows δ(id(G),x) = 1
proof -
  from A1 show thesis using
    group0_2_L1 monoid0.group0_1_L1 HomDiff_def id_conv group0_2_L6
    by simp
qed
```

The identity function $(I(x) = x)$ on G is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_4_L15: id(G) ∈ AH
```

```
proof -
  have G×G ≠ 0 using group0_2_L1 monoid0.group0_1_L3A
    by blast
  then show thesis using Group_ZF_3_4_L14 group0_2_L2
    id_type AlmostHoms_def by simp
qed
```

Almost homomorphisms form a monoid with composition. The identity function on the group is the neutral element there.

```
lemma (in group1) Group_ZF_3_4_L16:
  shows
  IsAmonoid(AH,Op2)
  monoid0(AH,Op2)
  id(G) = TheNeutralElement(AH,Op2)
proof-
  let i = TheNeutralElement(G→G,Composition(G))
  have
    IsAmonoid(G→G,Composition(G))
    monoid0(G→G,Composition(G))
    using monoid0_def Group_ZF_2_5_L2 by auto
  moreover have AH {is closed under} Composition(G)
    using Group_ZF_3_4_L6 by simp
  moreover have AH ⊆ G→G
    using AlmostHoms_def by auto
  moreover have i ∈ AH
    using Group_ZF_2_5_L2 Group_ZF_3_4_L15 by simp
  moreover have id(G) =  i
    using Group_ZF_2_5_L2 by simp
  ultimately show
    IsAmonoid(AH,Op2)
    monoid0(AH,Op2)
    id(G) = TheNeutralElement(AH,Op2)
    using monoid0.group0_1_T1 group0_1_L6 AlHomOp2_def monoid0_def
    by auto
qed
```

We can project the monoid of almost homomorphisms with composition to the group of almost homomorphisms divided by the subgroup of finite range functions. The class of the identity function is the neutral element of the quotient (monoid).

```
theorem (in group1) Group_ZF_3_4_T2:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  shows
  IsAmonoid(AH//R,ProjFun2(AH,R,Op2))
  R{id(G)} = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
proof -
  have group0(AH,Op1) using Group_ZF_3_2_L10A group0_def
    by simp
  with A1 groupAssum isAbelian show
```

```
    IsAmonoid(AH//R,ProjFun2(AH,R,Op2))
    R{id(G)} = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
    using Group_ZF_3_3_L2 group0.Group_ZF_2_4_L3 Group_ZF_3_4_L13A
      Group_ZF_3_4_L16 monoid0.Group_ZF_2_2_T1 Group_ZF_2_2_L1
    by auto
qed
```

16.5 Shifting almost homomorphisms

In this this section we consider what happens if we multiply an almost homomorphism by a group element. We show that the resulting function is also an a. h., and almost equal to the original one. This is used only for slopes (integer a.h.) in Int_ZF_2 where we need to correct a positive slopes by adding a constant, so that it is at least 2 on positive integers.

If s is an almost homomorphism and c is some constant from the group, then $s \cdot c$ is an almost homomorphism.

```
lemma (in group1) Group_ZF_3_5_L1:
  assumes A1: s ∈ AH and A2: c∈G and
  A3: r = {⟨x,s(x)·c⟩. x∈G}
  shows
  ∀x∈G. r(x) = s(x)·c
  r ∈ AH
  s ≈ r
proof -
  from A1 A2 A3 have I: r:G→G
    using AlmostHoms_def apply_funtype group_op_closed
    ZF_fun_from_total by auto
  with A3 show II: ∀x∈G. r(x) = s(x)·c
    using ZF_fun_from_tot_val by simp
  with isAbelian A1 A2 have III:
    ∀p ∈ G×G. δ(r,p) = δ(s,p)·c⁻¹
    using group_op_closed AlmostHoms_def apply_funtype
    HomDiff_def group0_4_L7 by auto
  have {δ(r,p). p ∈ G×G} ∈ Fin(G)
  proof -
    from A1 A2 have
      {δ(s,p). p ∈ G×G} ∈ Fin(G)   c⁻¹∈G
      using AlmostHoms_def inverse_in_group by auto
    then have {δ(s,p)·c⁻¹. p ∈ G×G} ∈ Fin(G)
      using group_oper_assocA Finite1_L16AA
      by simp
    moreover from III have
      {δ(r,p). p ∈ G×G} = {δ(s,p)·c⁻¹. p ∈ G×G}
      by (rule ZF1_1_L4B)
    ultimately show thesis by simp
  qed
  with I show IV: r ∈ AH using AlmostHoms_def
```

```
    by simp
  from isAbelian A1 A2 I II have
    ∀n ∈ G. s(n)·(r(n))⁻¹ = c⁻¹
    using AlmostHoms_def apply_funtype group0_4_L6AB
    by auto
  then have {s(n)·(r(n))⁻¹. n∈G} = {c⁻¹. n∈G}
    by (rule ZF1_1_L4B)
  with A1 A2 IV show s ≈ r
    using group0_2_L1 monoid0.group0_1_L3A
      inverse_in_group Group_ZF_3_4_L12 by simp
qed

end
```

17 OrderedGroup_ZF.thy

theory OrderedGroup_ZF **imports** Group_ZF Order_ZF Finite_ZF_1

begin

This theory file defines and shows the basic properties of (partially or linearly) ordered groups. We define the set of nonnegative elements and the absolute value function. 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.thy that subsets of integers are bounded iff they are finite.

17.1 Ordered groups

This section defines ordered groups.

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 $g \cdot a \leq g \cdot b$. We define the set of nonnegative elements in the obvious way as $G^+ = \{x \in G : 1 \leq x\}$. G_+ is a similar set, but without the unit. 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. We also define the maximum absolute value of a set, that is the maximum of the set $\{|x|.x \in A\}$. The odd functions are defined as those having property $f(a^{-1}) = (f(a))^{-1}$. Looks a bit strange in the multiplicative notation. For linearly oredered 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.

constdefs

```
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 ) )

Nonnegative(G,P,r) ≡ {x∈G. <TheNeutralElement(G,P),x> ∈ r}

PositiveSet(G,P,r) ≡
{x∈G. <TheNeutralElement(G,P),x> ∈ r ∧ TheNeutralElement(G,P)≠ x}

AbsoluteValue(G,P,r) ≡ id(Nonnegative(G,P,r)) ∪
restrict(GroupInv(G,P),G - Nonnegative(G,P,r))

OddExtension(G,P,r,f) ≡
(f ∪ {⟨a, GroupInv(G,P)(f(GroupInv(G,P)(a)))⟩.
a ∈ GroupInv(G,P)(PositiveSet(G,P,r))} ∪
{⟨TheNeutralElement(G,P),TheNeutralElement(G,P)⟩})
```

We will use a similar notation for ordered groups as for the generic groups. `G`$^+$ denotes the set of nonnegative elements (that satisfy $1 \leq a$ and `G`$_+$ is the set of (strictly) positive elements. `-A` is the set inverses of elements from A. I hope that using additive notation for this notion is not too shocking here. The symbol `f`$^\circ$ 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 nonnegative_def [simp]: G₊ ≡ PositiveSet(G,P,r)

  fixes setinv :: i⇒i (- _ 72)
  defines setninv_def [simp]: -A ≡ GroupInv(G,P)(A)

  fixes abs (| _ |)
  defines abs_def [simp]: |a| ≡ AbsoluteValue(G,P,r)(a)

  fixes oddext (_ °)
  defines oddext_def [simp]: f° ≡ OddExtension(G,P,r,f)
```

In `group3` context we can use the theorems proven in the `group0` context.

```
lemma (in group3) OrderedGroup_ZF_1_L1: shows group0(G,P)
  using ordGroupAssum IsAnOrdGroup_def group0_def by simp
```

Ordered group (carrier) is not empty. This is a property of monoids, but it is good to have it handy in the `group3` context.

```
lemma (in group3) OrderedGroup_ZF_1_L1A: shows G≠0
  using OrderedGroup_ZF_1_L1 group0.group0_2_L1 monoid0.group0_1_L3A
  by blast
```

The next lemma is just to see the definition of the nonnegative set in our notation.

```
lemma (in group3) OrderedGroup_ZF_1_L2:
  shows g∈G⁺ ⟷ 1≤g
  using ordGroupAssum IsAnOrdGroup_def Nonnegative_def
  by auto
```

The next lemma is just to see the definition of the positive set in our notation.

```
lemma (in group3) OrderedGroup_ZF_1_L2A:
  shows g∈G₊ ⟷ (1≤g ∧ g≠1)
  using ordGroupAssum IsAnOrdGroup_def PositiveSet_def
  by auto
```

For total order if g is not in G^+, then it has to be less or equal the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L2B:
  assumes A1: r {is total on} G and A2: a∈G-G⁺
  shows a≤1
proof -
  from A2 have a∈G ¬(1≤a) using OrderedGroup_ZF_1_L2 by auto
  with A1 show thesis
    using IsTotal_def OrderedGroup_ZF_1_L1 group0.group0_2_L2 by auto
qed
```

The group order is reflexive.

```
lemma (in group3) OrderedGroup_ZF_1_L3: assumes g∈G
  shows g≤g
  using ordGroupAssum prems IsAnOrdGroup_def IsPartOrder_def refl_def
  by simp
```

1 is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L3A: shows 1∈G⁺
  using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L3
    OrderedGroup_ZF_1_L1 group0.group0_2_L2 by simp
```

In this context $a \leq b$ implies that both a and b belong to G.

```
lemma (in group3) OrderedGroup_ZF_1_L4:
  assumes a≤b shows a∈G b∈G
  using ordGroupAssum prems IsAnOrdGroup_def by auto
```

It is good to have transitivity handy.

```
lemma (in group3) Group_order_transitive:
  assumes A1: a≤b  b≤c shows a≤c
proof -
```

```
  from ordGroupAssum have trans(r)
    using IsAnOrdGroup_def IsPartOrder_def
    by simp
  moreover from A1 have <a,b> ∈ r ∧ <b,c> ∈ r by simp
  ultimately have <a,c> ∈ r by (rule Fol1_L3)
  thus thesis by simp
qed
```

The order in an ordered group is antisymmetric.

```
lemma (in group3) group_order_antisym:
  assumes A1: a≤b  b≤a shows a=b
proof -
  from ordGroupAssum A1 have
    antisym(r)  <a,b> ∈ r  <b,a> ∈ r
    using IsAnOrdGroup_def IsPartOrder_def by auto
  then show a=b by (rule Fol1_L4)
qed
```

Transitivity for the strict order: if $a < b$ and $b \leq c$, then $a < c$.

```
lemma (in group3) OrderedGroup_ZF_1_L4A:
  assumes A1: a<b  and A2: b≤c
  shows a<c
proof -
  from A1 A2 have a≤b  b≤c by auto
  then have a≤c by (rule Group_order_transitive)
  moreover from A1 A2 have a≠c using group_order_antisym by auto
  ultimately show a<c by simp
qed
```

Another version of transitivity for the strict order: if $a \leq b$ and $b < c$, then $a < c$.

```
lemma (in group3) group_strict_ord_transit:
  assumes A1: a≤b and A2: b<c
  shows a<c
proof -
  from A1 A2 have a≤b  b≤c by auto
  then have  a≤c by (rule Group_order_transitive)
  moreover from A1 A2 have a≠c using group_order_antisym by auto
  ultimately show a<c by simp
qed
```

Strict order is preserved by translations.

```
lemma (in group3) group_strict_ord_transl_inv:
  assumes a<band c∈G
  shows
  a·c < b·c
  c·a < c·b
  using ordGroupAssum prems IsAnOrdGroup_def
```

```
    OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1 group0.group0_2_L19
  by auto
```

If the group order is total, then the group is ordered linearly.

```
lemma (in group3) group_ord_total_is_lin:
  assumes r {is total on} G
  shows IsLinOrder(G,r)
  using prems ordGroupAssum IsAnOrdGroup_def Order_ZF_1_L3
  by simp
```

For linearly ordered groups elements in the nonnegative set are greater than those in the complement.

```
lemma (in group3) OrderedGroup_ZF_1_L4B:
  assumes r {is total on} G
  and a∈G+ and b ∈ G-G+
  shows b≤a
proof -
  from prems have b≤1 1≤a
    using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L2B by auto
  thus thesis by (rule Group_order_transitive)
qed
```

If $a \leq 1$ and $a \neq 1$, then $a \in G \setminus G^+$.

```
lemma (in group3) OrderedGroup_ZF_1_L4C:
  assumes A1: a≤1 and A2: a≠1
  shows a ∈ G-G+
proof (rule ccontr)
  assume a ∉ G-G+
  with ordGroupAssum A1 A2 show False
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L2
     OrderedGroup_ZF_1_L4 IsAnOrdGroup_def IsPartOrder_def antisym_def
    by auto
qed
```

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 (rule ccontr)
  assume b ∉ G - G+
  with A2 have 1≤b b≤a
     using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L2 by auto
  then have 1≤a by (rule Group_order_transitive)
  with A1 show False using OrderedGroup_ZF_1_L2 by simp
qed
```

The nonnegative set is contained in the group.

```
lemma (in group3) OrderedGroup_ZF_1_L4E: shows G+ ⊆ G
```

```
  using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L4 by auto
```

Taking the inverse on both sides reverses the inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L5:
  assumes A1: a≤b shows b⁻¹≤a⁻¹
proof -
  from A1 have T1: a∈G b∈G a⁻¹∈G b⁻¹∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  with A1 ordGroupAssum have a·a⁻¹≤b·a⁻¹ using IsAnOrdGroup_def
    by simp
  with T1 ordGroupAssum have b⁻¹·1≤b⁻¹·(b·a⁻¹)
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6 IsAnOrdGroup_def
    by simp
  with T1 show thesis using
    OrderedGroup_ZF_1_L1 group0.group0_2_L2 group0.group_oper_assoc
    group0.group0_2_L6 by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5A:
  assumes A1: a≤1 shows 1≤a⁻¹
proof -
  from A1 have 1⁻¹≤a⁻¹ using OrderedGroup_ZF_1_L5
    by simp
  then show thesis using OrderedGroup_ZF_1_L1 group0.group_inv_of_one

    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5AA:
  assumes A1: a∈G and A2: 1≤a⁻¹
  shows a≤1
proof -
  from A2 have (a⁻¹)⁻¹≤1⁻¹ using OrderedGroup_ZF_1_L5
    by simp
  with A1 show a≤1
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv group0.group_inv_of_one
    by simp
qed
```

If an element is nonnegative, then the inverse is not greater that the unit. Also shows that nonnegative elements cannot be negative

```
lemma (in group3) OrderedGroup_ZF_1_L5AB:
  assumes A1: 1≤a shows a⁻¹≤1 and ¬(a≤1 ∧ a≠1)
proof -
```

```
  from A1 have a⁻¹≤1⁻¹
    using OrderedGroup_ZF_1_L5 by simp
  then show a⁻¹≤1 using OrderedGroup_ZF_1_L1 group0.group_inv_of_one
    by simp
  { assume a≤1 and a≠1
    with A1 have False using group_order_antisym
      by blast
  } then show ¬(a≤1 ∧ a≠1) by auto
qed
```

If two elements are greater or equal than the unit, then the inverse of one is not greater than the other.

```
lemma (in group3) OrderedGroup_ZF_1_L5AC:
  assumes A1: 1≤a  1≤b
  shows a⁻¹ ≤ b
proof -
  from A1 have a⁻¹≤1  1≤b
    using OrderedGroup_ZF_1_L5AB by auto
  then show a⁻¹ ≤ b by (rule Group_order_transitive)
qed
```

Taking negative on both sides reverses the inequality, case with an inverse on one side.

```
lemma (in group3) OrderedGroup_ZF_1_L5AD:
  assumes A1: b ∈ G and A2: a≤b⁻¹
  shows b ≤ a⁻¹
proof -
  from A2 have (b⁻¹)⁻¹ ≤ a⁻¹
    using OrderedGroup_ZF_1_L5 by simp
  with A1 show b ≤ a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5AE:
  assumes A1: a∈G  b∈G  c∈G and A2: a·b ≤ a·c
  shows b≤c
proof -
  from ordGroupAssum A1 A2 have a⁻¹·(a·b) ≤ a⁻¹·(a·c)
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      IsAnOrdGroup_def by simp
  with A1 show b≤c
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by simp
qed
```

We can cancel the same element on both sides of an inequality, a version with an inverse on both sides.

```
lemma (in group3) OrderedGroup_ZF_1_L5AF:
  assumes A1: a∈G  b∈G  c∈G and A2: a·b⁻¹ ≤ a·c⁻¹
  shows c≤b
proof -
  from A1 A2 have (c⁻¹)⁻¹ ≤ (b⁻¹)⁻¹
     using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      OrderedGroup_ZF_1_L5AE OrderedGroup_ZF_1_L5 by simp
  with A1 show c≤b
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv by simp
qed
```

Taking negative on both sides reverses the inequality, another case with an inverse on one side.

```
lemma (in group3) OrderedGroup_ZF_1_L5AG:
  assumes A1: a ∈ G and A2: a⁻¹≤b
  shows b⁻¹ ≤ a
proof -
  from A2 have b⁻¹ ≤ (a⁻¹)⁻¹
    using OrderedGroup_ZF_1_L5 by simp
  with A1 show b⁻¹ ≤ a
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

We can multiply the sides of two inequalities.

```
lemma (in group3) OrderedGroup_ZF_1_L5B:
  assumes A1: a≤b and A2: c≤d
  shows a·c ≤ b·d
proof -
  from A1 A2 have c∈G b∈G using OrderedGroup_ZF_1_L4 by auto
  with A1 A2 ordGroupAssum have a·c≤ b·c b·c≤b·d
    using IsAnOrdGroup_def by auto
  then show a·c ≤ b·d by (rule Group_order_transitive)
qed
```

We can replace first of the factors on one side of an inequality with a greater one.

```
lemma (in group3) OrderedGroup_ZF_1_L5C:
  assumes A1: c∈G and A2: a≤b·c and A3: b≤b₁
  shows a≤b₁·c
proof -
  from A1 A3 have b·c ≤ b₁·c
    using OrderedGroup_ZF_1_L3 OrderedGroup_ZF_1_L5B by simp
  with A2 show a≤b₁·c by (rule Group_order_transitive)
qed
```

We can replace second of the factors on one side of an inequality with a greater one.

```
lemma (in group3) OrderedGroup_ZF_1_L5D:
  assumes A1: b∈G and A2: a ≤ b·c and A3: c≤b₁
  shows a ≤ b·b₁
proof -
  from A1 A3 have b·c ≤ b·b₁
    using OrderedGroup_ZF_1_L3 OrderedGroup_ZF_1_L5B by auto
  with A2 show a≤b·b₁ by (rule Group_order_transitive)
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5E:
  assumes A1: a ≤ b·c and A2: b≤b₁  c≤c₁
  shows a ≤ b₁·c₁
proof -
  from A2 have b·c ≤ b₁·c₁ using OrderedGroup_ZF_1_L5B
    by simp
  with A1 show a≤b₁·c₁ by (rule Group_order_transitive)
qed
```

We don't decrease an element of the group by multiplying by one that is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L5F:
  assumes A1: 1≤a and A2: b∈G
  shows b≤a·b  b≤b·a
proof -
  from ordGroupAssum A1 A2 have
    1·b≤a·b  b·1≤b·a
    using IsAnOrdGroup_def by auto
  with A2 show b≤a·b  b≤b·a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5G: assumes A1: a≤b
  and A2: 1≤c shows a≤b·c  a≤c·b
proof -
  from A1 A2 have I: b≤b·c  and II: b≤c·b
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L5F by auto
  from A1 I show a≤b·c by (rule Group_order_transitive)
  from A1 II show a≤c·b by (rule Group_order_transitive)
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5H:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ≤ c
  shows
```

```
  a ≤ c·b
  c⁻¹·a ≤ b
proof -
  from A2 have T: c∈G  c⁻¹ ∈ G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  from ordGroupAssum A1 A2 have a·b⁻¹·b ≤ c·b
    using IsAnOrdGroup_def by simp
  with A1 show a ≤ c·b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by simp
  with ordGroupAssum A2 T have c⁻¹·a ≤ c⁻¹·(c·b)
    using IsAnOrdGroup_def by simp
  with A1 T show c⁻¹·a ≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L5I:
  assumes a≤b and c≤d
  shows a·d⁻¹ ≤ b·c⁻¹
  using prems OrderedGroup_ZF_1_L5 OrderedGroup_ZF_1_L5B
  by simp
```

We can put an element on the other side of an inequality changing its sign, version with the inverse.

```
lemma (in group3) OrderedGroup_ZF_1_L5J:
  assumes A1: a∈G  b∈G and A2: c ≤ a·b⁻¹
  shows c·b ≤ a
proof -
  from ordGroupAssum A1 A2 have c·b ≤ a·b⁻¹·b
    using IsAnOrdGroup_def by simp
  with A1 show c·b ≤ a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by simp
qed
```

We can put an element on the other side of an inequality changing its sign, version with the inverse.

```
lemma (in group3) OrderedGroup_ZF_1_L5JA:
  assumes A1: a∈G  b∈G and A2: c ≤ a⁻¹·b
  shows a·c≤ b
proof -
  from ordGroupAssum A1 A2 have a·c ≤ a·(a⁻¹·b)
    using IsAnOrdGroup_def by simp
  with A1 show a·c≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
```

```
    by simp
qed
```

A special case of `OrderedGroup_ZF_1_L5J` where $c = 1$.

```
corollary (in group3) OrderedGroup_ZF_1_L5K:
  assumes A1: a∈G  b∈G and A2: 1 ≤ a·b⁻¹
  shows b ≤ a
proof -
  from A1 A2 have 1·b ≤ a
    using OrderedGroup_ZF_1_L5J by simp
  with A1 show b ≤ a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

A special case of `OrderedGroup_ZF_1_L5JA` where $c = 1$.

```
corollary (in group3) OrderedGroup_ZF_1_L5KA:
  assumes A1: a∈G  b∈G and A2: 1 ≤ a⁻¹·b
  shows a ≤ b
proof -
  from A1 A2 have a·1 ≤ b
    using OrderedGroup_ZF_1_L5JA by simp
  with A1 show a ≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

If the order is total, the elements that do not belong to the positive set are negative. We also show here that the group inverse of an element that does not belong to the nonnegative set does belong to the nonnegative set.

```
lemma (in group3) OrderedGroup_ZF_1_L6:
  assumes A1: r {is total on} G and A2: a∈G-G⁺
  shows a≤1  a⁻¹ ∈ G⁺  restrict(GroupInv(G,P),G-G⁺)(a) ∈ G⁺
proof -
  from A2 have T1: a∈G a∉G⁺ 1∈G
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2 by auto
  with A1 show a≤1 using OrderedGroup_ZF_1_L2 IsTotal_def
    by auto
  then show a⁻¹ ∈ G⁺ using OrderedGroup_ZF_1_L5A OrderedGroup_ZF_1_L2
    by simp
  with A2 show restrict(GroupInv(G,P),G-G⁺)(a) ∈ G⁺
    using restrict by simp
qed
```

If a property is invariant with respect to taking the inverse and it is true on the nonnegative set, than it is true on the whole group.

```
lemma (in group3)  OrderedGroup_ZF_1_L7:
  assumes A1: r {is total on} G
```

```
  and A2: ∀a∈G⁺.∀b∈G⁺. Q(a,b)
  and A3: ∀a∈G.∀b∈G. Q(a,b)⟶Q(a⁻¹,b)
  and A4: ∀a∈G.∀b∈G. Q(a,b)⟶Q(a,b⁻¹)
  and A5: a∈G b∈G
  shows Q(a,b)
proof (cases a∈G⁺)
  assume A6: a∈G⁺ show Q(a,b)
  proof (cases b∈G⁺)
    assume b∈G⁺
    with A6 A2 show Q(a,b) by simp
  next assume b∉G⁺
    with A1 A2 A4 A5 A6 have Q(a,(b⁻¹)⁻¹)
      using OrderedGroup_ZF_1_L6 OrderedGroup_ZF_1_L1 group0.inverse_in_group
      by simp
    with A5 show Q(a,b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp
  qed
next assume a∉G⁺
  with A1 A5 have T1: a⁻¹ ∈ G⁺ using OrderedGroup_ZF_1_L6 by simp
  show Q(a,b)
  proof (cases b∈G⁺)
    assume b∈G⁺
    with A2 A3 A5 T1 have Q((a⁻¹)⁻¹,b)
      using OrderedGroup_ZF_1_L1 group0.inverse_in_group by simp
    with A5 show Q(a,b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp
  next assume b∉G⁺
    with A1 A2 A3 A4 A5 T1 have Q((a⁻¹)⁻¹,(b⁻¹)⁻¹)
       using OrderedGroup_ZF_1_L6 OrderedGroup_ZF_1_L1 group0.inverse_in_group
       by simp
    with A5 show Q(a,b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp
  qed
qed
```

A lemma about splitting the ordered group "plane" into 6 subsets. Useful for proofs by cases.

```
lemma (in group3) OrdGroup_6cases: assumes A1: r {is total on} G
  and A2:  a∈G  b∈G
  shows
  1≤a ∧ 1≤b  ∨  a≤1 ∧ b≤1  ∨
  a≤1 ∧ 1≤b ∧ 1 ≤ a·b  ∨ a≤1 ∧ 1≤b ∧ a·b ≤ 1  ∨
  1≤a ∧ b≤1 ∧ 1 ≤ a·b  ∨  1≤a ∧ b≤1 ∧ a·b ≤ 1
proof -
  from A1 A2 have
    1≤a ∨ a≤1
    1≤b ∨ b≤1
    1 ≤ a·b ∨ a·b ≤ 1
    using OrderedGroup_ZF_1_L1 group0.group_op_closed group0.group0_2_L2
```

```
      IsTotal_def by auto
  then show thesis by auto
qed
```

The next lemma shows what happens when one element of a totally ordered group is not greater or equal than another.

```
lemma (in group3) OrderedGroup_ZF_1_L8:
  assumes A1: r {is total on} G
  and A2: a∈G  b∈G
  and A3: ¬(a≤b)
  shows b ≤ a  a⁻¹ ≤ b⁻¹  a≠b  b<a

proof -
  from A1 A2 A3 show I: b ≤ a using IsTotal_def
    by auto
  then show a⁻¹ ≤ b⁻¹ using OrderedGroup_ZF_1_L5 by simp
  from A2 have a ≤ a using OrderedGroup_ZF_1_L3 by simp
  with I A3 show a≠b  b < a by auto
qed
```

If one element is greater or equal and not equal to another, then it is not smaller or equal.

```
lemma (in group3) OrderedGroup_ZF_1_L8AA:
  assumes A1: a≤b and A2: a≠b
  shows ¬(b≤a)
proof -
  { note A1
    moreover assume b≤a
    ultimately have a=b by (rule group_order_antisym)
    with A2 have False by simp
  } thus ¬(b≤a) by auto
qed
```

A special case of `OrderedGroup_ZF_1_L8` when one of the elements is the unit.

```
corollary (in group3) OrderedGroup_ZF_1_L8A:
  assumes A1: r {is total on} G
  and A2: a∈G and A3: ¬(1≤a)
  shows 1 ≤ a⁻¹  1≠a  a≤1
proof -
  from A1 A2 A3 have I:
    r {is total on} G
    1∈G  a∈G
     ¬(1≤a)
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  then have 1⁻¹ ≤ a⁻¹
    by (rule OrderedGroup_ZF_1_L8)
  then show 1 ≤ a⁻¹
```

```
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_one by simp
  from I show 1≠a by (rule OrderedGroup_ZF_1_L8)
  from A1 I show a≤1 using IsTotal_def
    by auto
qed
```

A negative element can not be nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L8B:
  assumes A1: a≤1  and A2: a≠1 shows ¬(1≤a)
proof -
  { assume  1≤a
    with A1 have a=1 using group_order_antisym
      by auto
    with A2 have False by simp
  } thus thesis by auto
qed
```

An element is greater or equal than another iff the difference is nonpositive.

```
lemma (in group3) OrderedGroup_ZF_1_L9:
  assumes A1: a∈G  b∈G
  shows a≤b ⟷ a·b⁻¹ ≤ 1
proof
  assume a ≤ b
  with ordGroupAssum A1 have a·b⁻¹ ≤ b·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group
    IsAnOrdGroup_def by simp
  with A1 show a·b⁻¹ ≤ 1
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6
    by simp
next assume A2: a·b⁻¹ ≤ 1
  with ordGroupAssum A1 have a·b⁻¹·b ≤ 1·b
    using IsAnOrdGroup_def by simp
  with A1 show a ≤ b
    using OrderedGroup_ZF_1_L1
      group0.group0_2_L16 group0.group0_2_L2
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L9A:
  assumes A1: a∈G  b∈G  c∈G
  shows a·b ≤ c  ⟷ a ≤ c·b⁻¹
proof
  assume a·b ≤ c
  with ordGroupAssum A1 have a·b·b⁻¹ ≤ c·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def
    by simp
  with A1 show a ≤ c·b⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16 by simp
```

```
next assume a ≤ c·b⁻¹
  with ordGroupAssum A1 have a·b ≤ c·b⁻¹·b
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def
    by simp
  with A1 show a·b ≤ c
     using OrderedGroup_ZF_1_L1 group0.group0_2_L16 by simp
qed
```

A one side version of the previous lemma with weaker assuptions.

```
lemma (in group3) OrderedGroup_ZF_1_L9B:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ≤ c
  shows a ≤ c·b
proof -
  from A1 A2 have a∈G  b⁻¹∈G  c∈G
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      OrderedGroup_ZF_1_L4 by auto
  with A1 A2 show a ≤ c·b
    using OrderedGroup_ZF_1_L9A OrderedGroup_ZF_1_L1
      group0.group_inv_of_inv by simp
qed
```

We can put en element on the other side of inequality, changing its sign.

```
lemma (in group3) OrderedGroup_ZF_1_L9C:
  assumes A1: a∈G  b∈G and A2: c≤a·b
  shows
  c·b⁻¹ ≤ a
  a⁻¹·c ≤ b
proof -
  from ordGroupAssum A1 A2 have
    c·b⁻¹ ≤ a·b·b⁻¹
    a⁻¹·c ≤ a⁻¹·(a·b)
    using OrderedGroup_ZF_1_L1 group0.inverse_in_group IsAnOrdGroup_def
    by auto
  with A1 show
    c·b⁻¹ ≤ a
    a⁻¹·c ≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by auto
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L9D: assumes A1: a≤b
  shows 1 ≤ b·a⁻¹
proof -
  from A1 have T: a∈G  b∈G   a⁻¹ ∈ G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  with ordGroupAssum A1 have a·a⁻¹ ≤ b·a⁻¹
```

```
    using IsAnOrdGroup_def by simp
  with T show 1 ≤ b·a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L9E:
  assumes A1: a≤b  a≠b
  shows 1 ≤ b·a⁻¹  1 ≠ b·a⁻¹  b·a⁻¹ ∈ G₊
proof -
  from A1 have T: a∈G  b∈G using OrderedGroup_ZF_1_L4
    by auto
  from A1 show I: 1 ≤ b·a⁻¹ using OrderedGroup_ZF_1_L9D
    by simp
  { assume b·a⁻¹ = 1
    with T have a=b
      using OrderedGroup_ZF_1_L1 group0.group0_2_L11A
      by auto
    with A1 have False by simp
  } then show 1 ≠ b·a⁻¹ by auto
  then have b·a⁻¹ ≠ 1 by auto
  with I show b·a⁻¹ ∈ G₊ using OrderedGroup_ZF_1_L2A
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L9F:
  assumes A1: a∈G  b∈G and A2: 1 ≤ b·a⁻¹
  shows a≤b
proof -
  from A1 A2 have 1·a ≤ b
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L9A
    by simp
  with A1 show a≤b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L10:
  assumes a∈G  b∈G and c≤d
  shows a·c·b ≤ a·d·b
  using ordGroupAssum prems IsAnOrdGroup_def by simp
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L11:
  assumes A1: 1≤a  1≤b
```

```
  and A2: 1 ≠ a  1 ≠ b
  shows 1 ≠ a·b
proof -
  from A1 have T1: a∈G  b∈G
    using OrderedGroup_ZF_1_L4 by auto
  { assume 1 = a·b
    with A1 T1 have a≤1  1≤a
      using OrderedGroup_ZF_1_L1 group0.group0_2_L9 OrderedGroup_ZF_1_L5AA

      by auto
    then have a = 1 by (rule group_order_antisym)
    with A2 have False by simp
  } then show 1 ≠ a·b by auto
qed
```

A product of nonnegative elements is nonnegative.

```
lemma (in group3) OrderedGroup_ZF_1_L12:
  assumes A1: 1 ≤ a  1 ≤ b
  shows 1 ≤ a·b
proof -
  from A1 have 1·1 ≤ a·b
    using  OrderedGroup_ZF_1_L5B by simp
  then show 1 ≤ a·b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
qed
```

If a is not greater than b, then 1 is not greater than $b \cdot a^{-1}$.

```
lemma (in group3) OrderedGroup_ZF_1_L12A:
  assumes A1: a≤b shows 1 ≤ b·a⁻¹
proof -
  from A1 have T: 1 ∈ G  a∈G  b∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  with A1 have 1·a ≤ b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
  with T show 1 ≤ b·a⁻¹ using OrderedGroup_ZF_1_L9A
    by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L12B:
  assumes A1: a∈G  b∈G and  A2: a·b⁻¹ < c
  shows a < c·b
proof -
  from A1 A2 have a·b⁻¹·b < c·b
    using group_strict_ord_transl_inv by auto
  moreover from A1 have a·b⁻¹·b = a
```

```
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by simp
  ultimately show a < c·b
    by auto
qed
```

We can multiply the sides of two inequalities, first of them strict and we get a strict inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L12C:
  assumes A1: a<b and A2: c≤d
  shows a·c < b·d
proof -
  from A1 A2 have T: a∈G  b∈G  c∈G  d∈G
    using OrderedGroup_ZF_1_L4 by auto
  with ordGroupAssum A2 have a·c ≤ a·d
    using IsAnOrdGroup_def by simp
  moreover from A1 T have a·d < b·d
    using group_strict_ord_transl_inv by simp
  ultimately show a·c < b·d
    by (rule group_strict_ord_transit)
qed
```

We can multiply the sides of two inequalities, second of them strict and we get a strict inequality.

```
lemma (in group3) OrderedGroup_ZF_1_L12D:
  assumes A1: a≤b and A2: c<d
  shows a·c < b·d
proof -
  from A1 A2 have T: a∈G  b∈G  c∈G  d∈G
    using OrderedGroup_ZF_1_L4 by auto
  with A2 have a·c < a·d
    using group_strict_ord_transl_inv by simp
  moreover from ordGroupAssum A1 T have a·d ≤ b·d
     using IsAnOrdGroup_def by simp
   ultimately show a·c < b·d
     by (rule OrderedGroup_ZF_1_L4A)
qed
```

17.2 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: G+ {is closed under} P
proof -
  { fix a b assume a∈G+  b∈G+
    then have T1: 1 ≤ a·b   and 1 ≠ a·b
      using PositiveSet_def OrderedGroup_ZF_1_L11 OrderedGroup_ZF_1_L12
      by auto
    moreover from T1 have a·b ∈ G
      using OrderedGroup_ZF_1_L4 by simp
    ultimately have a·b ∈ G+ using PositiveSet_def by simp
  } then show G+ {is closed under} P using IsOpClosed_def
    by simp
qed
```

For totally ordered groups every nonunit element is positive or its inverse is positive.

```
lemma (in group3) OrderedGroup_ZF_1_L14:
  assumes A1: r {is total on} G and A2: a∈G
  shows a=1 ∨ a∈G+ ∨ a⁻¹∈G+
proof -
  { assume A3: a≠1
    moreover from A1 A2 have a≤1 ∨ 1≤a
      using IsTotal_def OrderedGroup_ZF_1_L1 group0.group0_2_L2
      by simp
    moreover from A3 A2 have T1: a⁻¹ ≠ 1
      using OrderedGroup_ZF_1_L1 group0.group0_2_L8B
      by simp
    ultimately have a⁻¹∈G+ ∨ a∈G+
      using OrderedGroup_ZF_1_L5A OrderedGroup_ZF_1_L2A
      by auto
  } thus a=1 ∨ a∈G+ ∨ a⁻¹∈G+ by auto
qed
```

If an element belongs to the positive set, then it is not the unit and its inverse does not belong to the positive set.

```
lemma (in group3) OrderedGroup_ZF_1_L15:
   assumes A1: a∈G+  shows a≠1  a⁻¹∉G+
proof -
  from A1 show T1: a≠1 using PositiveSet_def by auto
  { assume a⁻¹ ∈ G+
    with A1 have a≤1  1≤a
      using OrderedGroup_ZF_1_L5AA PositiveSet_def by auto
    then have a=1 by (rule group_order_antisym)
    with T1 have False by simp
  } then show a⁻¹∉G+ by auto
qed
```

If a^{-1} is positive, then a can not be positive or the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L16:
```

```
  assumes A1: a∈G and A2: a⁻¹∈G₊ shows a≠1  a∉G₊
proof -
  from A2 have a⁻¹≠1  (a⁻¹)⁻¹ ∉ G₊
    using OrderedGroup_ZF_1_L15 by auto
  with A1 show a≠1  a∉G₊
    using OrderedGroup_ZF_1_L1 group0.group0_2_L8C group0.group_inv_of_inv

    by auto
qed
```

For linearly ordered groups each element is either the unit, positive or its inverse is positive.

```
lemma (in group3) OrdGroup_decomp:
  assumes A1: r {is total on} G and A2: a∈G
  shows Exactly_1_of_3_holds (a=1,a∈G₊,a⁻¹∈G₊)
proof -
  from A1 A2 have a=1 ∨ a∈G₊ ∨ a⁻¹∈G₊
    using OrderedGroup_ZF_1_L14 by simp
  moreover from A2 have a=1 ⟶ (a∉G₊ ∧ a⁻¹∉G₊)
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_one
    PositiveSet_def by simp
  moreover from A2 have a∈G₊ ⟶ (a≠1 ∧ a⁻¹∉G₊)
    using OrderedGroup_ZF_1_L15 by simp
  moreover from A2 have a⁻¹∈G₊ ⟶ (a≠1 ∧ a∉G₊)
    using OrderedGroup_ZF_1_L16 by simp
  ultimately show Exactly_1_of_3_holds (a=1,a∈G₊,a⁻¹∈G₊)
    by (rule Fol1_L5)
qed
```

A if a is a nonunit element that is not positive, then a^{-1} is is positive. This is useful for some proofs by cases.

```
lemma (in group3) OrdGroup_cases:
  assumes A1: r {is total on} G and A2: a∈G
  and A3: a≠1  a∉G₊
  shows a⁻¹ ∈ G₊
proof -
  from A1 A2 have a=1 ∨ a∈G₊ ∨ a⁻¹∈G₊
    using OrderedGroup_ZF_1_L14 by simp
  with A3 show a⁻¹ ∈ G₊ by auto
qed
```

Elements from $G \setminus G_+$ are not greater that the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L17:
  assumes A1: r {is total on} G and A2: a ∈ G-G₊
  shows a≤1
proof (cases a = 1)
  assume a=1
  with A2 show a≤1 using OrderedGroup_ZF_1_L3 by simp
```

```
next assume a≠1
  with A1 A2 show a≤1
    using PositiveSet_def OrderedGroup_ZF_1_L8A
    by auto
qed
```

The next lemma allows to split proofs that something holds for all $a \in G$ into cases $a = 1$, $a \in G_+$, $-a \in G_+$.

```
lemma (in group3) OrderedGroup_ZF_1_L18:
  assumes A1: r {is total on} G and A2: b∈G
  and A3: Q(1) and A4: ∀a∈G+. Q(a) and A5: ∀a∈G+. Q(a⁻¹)
  shows Q(b)
proof -
  from A1 A2 A3 A4 A5 have Q(b) ∨ Q((b⁻¹)⁻¹)
    using OrderedGroup_ZF_1_L14 by auto
  with A2 show Q(b) using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by simp
qed
```

All elements greater or equal than an element of G_+ belong to G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L19:
  assumes A1: a ∈ G+ and A2: a≤b
  shows b ∈ G+
proof -
  from A1 have I: 1≤a  and II: a≠1
    using OrderedGroup_ZF_1_L2A by auto
  from I A2 have 1≤b by (rule Group_order_transitive)
  moreover have b≠1
  proof -
    { assume b=1
      with I A2 have 1≤a  a≤1
        by auto
      then have 1=a by (rule group_order_antisym)
      with II have False by simp
    } then show b≠1 by auto
  qed
  ultimately show b ∈ G+
    using OrderedGroup_ZF_1_L2A by simp
qed
```

The inverse of an element of G_+ cannot be in G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L20:
  assumes A1: r {is total on} G and A2: a ∈ G+
  shows a⁻¹ ∉ G+
proof -
  from A2 have a∈G using PositiveSet_def
    by simp
  with A1 have Exactly_1_of_3_holds (a=1,a∈G+,a⁻¹∈G+)
    using OrdGroup_decomp by simp
```

```
  with A2 show a⁻¹ ∉ G₊ by (rule Fol1_L7)
qed
```

The set of positive elements of a nontrivial linearly ordered group is not empty.

```
lemma (in group3) OrderedGroup_ZF_1_L21:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  shows G₊ ≠ 0
proof -
  have 1 ∈ G using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
  with A2 obtain a where a∈G  a≠1 by auto
  with A1 have a∈G₊ ∨ a⁻¹∈G₊
    using OrderedGroup_ZF_1_L14 by auto
  then show G₊ ≠ 0 by auto
qed
```

If b ∈G₊, then $a < a \cdot b$. Multiplying a by a positive elemnt increases a.

```
lemma (in group3) OrderedGroup_ZF_1_L22:
  assumes A1: a∈G  b∈G₊
  shows a≤a·b  a ≠ a·b  a·b ∈ G
proof -
  from ordGroupAssum A1 have a·1 ≤ a·b
    using OrderedGroup_ZF_1_L2A IsAnOrdGroup_def
    by simp
  with A1 show a≤a·b
    using OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by simp
  then show a·b ∈ G
    using OrderedGroup_ZF_1_L4 by simp
  { from A1 have a∈G  b∈G
      using PositiveSet_def by auto
    moreover assume a = a·b
    ultimately have b = 1
      using OrderedGroup_ZF_1_L1 group0.group0_2_L7
      by simp
    with A1 have False using PositiveSet_def
      by simp
  } then show a ≠ a·b by auto
qed
```

If G is a nontrivial linearly ordered hroup, then for every element of G we can find one in G₊ that is greater or equal.

```
lemma (in group3) OrderedGroup_ZF_1_L23:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  and A3: a∈G
  shows ∃b∈G₊. a≤b
proof (cases a∈G₊)
```

```
  assume A4: a∈G₊ then have a≤a
    using PositiveSet_def OrderedGroup_ZF_1_L3
    by simp
  with A4 show ∃b∈G₊. a≤b by auto
next assume a∉G₊
  with A1 A3 have I: a≤1 using OrderedGroup_ZF_1_L17
    by simp
  from A1 A2 obtain b where II: b∈G₊
    using OrderedGroup_ZF_1_L21 by auto
  then have 1≤b using PositiveSet_def by simp
  with I have a≤b by (rule Group_order_transitive)
  with II show ∃b∈G₊. a≤b by auto
qed
```

The G^+ is G_+ plus the unit.

```
lemma (in group3) OrderedGroup_ZF_1_L24: shows G⁺ = G₊∪{1}
  using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_1_L2A OrderedGroup_ZF_1_L3A
  by auto
```

What is $-G_+$, really?

```
lemma (in group3) OrderedGroup_ZF_1_L25: shows
  (-G₊) = {a⁻¹. a∈G₊}
  (-G₊) ⊆ G
proof -
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  moreover have G₊ ⊆ G using PositiveSet_def by auto
  ultimately show
    (-G₊) = {a⁻¹. a∈G₊}
    (-G₊) ⊆ G
    using func_imagedef func1_1_L6 by auto
qed
```

If the inverse of a is in G_+, then a is in the inverse of G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L26:
  assumes A1: a∈G and A2: a⁻¹ ∈ G₊
  shows a ∈ (-G₊)
proof -
  from A1 have a⁻¹ ∈ G  a = (a⁻¹)⁻¹ using OrderedGroup_ZF_1_L1
    group0.inverse_in_group group0.group_inv_of_inv
    by auto
  with A2 show a ∈ (-G₊) using OrderedGroup_ZF_1_L25
    by auto
qed
```

If a is in the inverse of G_+, then its inverse is in G_+.

```
lemma (in group3) OrderedGroup_ZF_1_L27:
  assumes a ∈ (-G₊)
```

```
  shows a⁻¹ ∈ G₊
  using prems OrderedGroup_ZF_1_L25 PositiveSet_def
    OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
  by auto
```

A linearly ordered group can be decomposed into G_+, $\{1\}$ and $-G$

```
lemma (in group3) OrdGroup_decomp2:
  assumes A1: r {is total on} G
  shows
  G = G₊ ∪ (-G₊)∪ {1}
  G₊∩(-G₊) = 0
  1 ∉ G₊∪(-G₊)
proof -
  { fix a assume A2: a∈G
    with A1 have a∈G₊ ∨ a⁻¹∈G₊ ∨ a=1
      using OrderedGroup_ZF_1_L14 by auto
    with A2 have a∈G₊ ∨ a∈(-G₊) ∨ a=1
      using OrderedGroup_ZF_1_L26 by auto
    then have a ∈ (G₊ ∪ (-G₊)∪ {1})
      by auto
  } then have G ⊆ G₊ ∪ (-G₊)∪ {1}
    by auto
  moreover have G₊ ∪ (-G₊)∪ {1} ⊆ G
    using OrderedGroup_ZF_1_L25 PositiveSet_def
      OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  ultimately show G = G₊ ∪ (-G₊)∪ {1} by auto
  { def DA: A ≡ G₊∩(-G₊)
    assume G₊∩(-G₊) ≠ 0
    with DA have A≠0 by simp
    then obtain a where a∈A by auto
    with DA have False using OrderedGroup_ZF_1_L15 OrderedGroup_ZF_1_L27
      by auto
  } then show G₊∩(-G₊) = 0 by auto
  show 1 ∉ G₊∪(-G₊)
    using OrderedGroup_ZF_1_L27
      OrderedGroup_ZF_1_L1 group0.group_inv_of_one
      OrderedGroup_ZF_1_L2A by auto
qed
```

If $a \cdot b^{-1}$ is nonnegative, then $b \leq a$. This maybe used to recover the order from the set of nonnegative elements and serve as a way to define order by prescibing that set (see the "Alternative definitions" section.

```
lemma (in group3) OrderedGroup_ZF_1_L28:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ∈ G⁺
  shows b≤a
proof -
  from A2 have 1 ≤ a·b⁻¹ using OrderedGroup_ZF_1_L2
    by simp
```

```
  with A1 show b≤a using OrderedGroup_ZF_1_L5K
    by simp
qed
```

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

```
corollary (in group3) OrderedGroup_ZF_1_L29:
  assumes A1: a∈G  b∈G and A2: a·b⁻¹ ∈ G₊
  shows b≤a  b≠a
proof -
  from A2 have 1 ≤ a·b⁻¹ and I: a·b⁻¹ ≠ 1
    using OrderedGroup_ZF_1_L2A by auto
  with A1 show b≤a using OrderedGroup_ZF_1_L5K
    by simp
  from A1 I show b≠a
    using OrderedGroup_ZF_1_L1 group0.group0_2_L6
    by auto
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_1_L30:
  assumes a∈G  b∈G and a=b ∨ b·a⁻¹ ∈ G₊
  shows a≤b
  using prems OrderedGroup_ZF_1_L3 OrderedGroup_ZF_1_L29
  by auto
```

A different take on decomposition: we can have $a = b$ or $a < b$ or $b < a$.

```
lemma (in group3) OrderedGroup_ZF_1_L31:
  assumes A1: r {is total on} G and A2: a∈G  b∈G
  shows a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
proof -
  from A2 have a·b⁻¹ ∈ G using OrderedGroup_ZF_1_L1
    group0.inverse_in_group group0.group_op_closed
    by simp
  with A1 have a·b⁻¹ = 1 ∨ a·b⁻¹ ∈ G₊ ∨ (a·b⁻¹)⁻¹ ∈ G₊
    using OrderedGroup_ZF_1_L14 by simp
  moreover
  { assume a·b⁻¹ = 1
    then have a·b⁻¹·b = 1·b by simp
    with A2 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup_ZF_1_L1
        group0.group0_2_L16 group0.group0_2_L2 by auto }
  moreover
  { assume a·b⁻¹ ∈ G₊
    with A2 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup_ZF_1_L29 by auto }
  moreover
  { assume (a·b⁻¹)⁻¹ ∈ G₊
    with A2 have b·a⁻¹ ∈ G₊ using OrderedGroup_ZF_1_L1
```

```
      group0.group0_2_L12 by simp
    with A2 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
      using OrderedGroup_ZF_1_L29 by auto }
  ultimately show a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
    by auto
qed
```

17.3 Intervals and bounded sets

A bounded set can be translated to put it in G^+ and then it is still bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L1:
  assumes A1: ∀g∈A. L≤g ∧ g≤M
  and A2: S = RightTranslation(G,P,L⁻¹)
  and A3: a ∈ S(A)
  shows a ≤ M·L⁻¹   1≤a
proof -
  from A3 have A≠0 using func1_1_L13A by fast
  then obtain g where g∈A by auto
  with A1 have T1: L∈G M∈G L⁻¹∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
    group0.inverse_in_group by auto
  with A2 have S : G→G using OrderedGroup_ZF_1_L1 group0.group0_5_L1
    by simp
  moreover from A1 have T2: A⊆G using OrderedGroup_ZF_1_L4 by auto
  ultimately have S(A) = {S(b). b∈A} using func_imagedef
    by simp
  with A3 obtain b where T3: b∈A a = S(b) by auto
  with A1 ordGroupAssum T1 have b·L⁻¹≤M·L⁻¹ L·L⁻¹≤b·L⁻¹
    using IsAnOrdGroup_def by auto
  with T3 A2 T1 T2 show a≤M·L⁻¹ 1≤a
    using OrderedGroup_ZF_1_L1 group0.group0_5_L2 group0.group0_2_L6
    by auto
qed
```

Every bounded set is an image of a subset of an interval that starts at 1.

```
lemma (in group3) OrderedGroup_ZF_2_L2:
  assumes A1: IsBounded(A,r)
  shows ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
proof (cases A=0)
  assume A2: A=0
  let B = 0
  let g = 1
  let T = ConstantFunction(G,1)
  have g∈G⁺ using OrderedGroup_ZF_1_L3A by simp
  moreover have T : G→G
    using func1_3_L1 OrderedGroup_ZF_1_L1 group0.group0_2_L2 by simp
  moreover from A2 have A = T(B) by simp
```

```
  moreover have B ⊆ Interval(r,1,g) by simp
  ultimately show
    ∃B.∃g∈G+.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
    by auto
next assume A3: A≠0
  with A1 obtain L U where D1: ∀x∈A. L≤x ∧ x≤U
    using IsBounded_def IsBoundedBelow_def IsBoundedAbove_def
    by auto
  with A3 have T1: A⊆G using OrderedGroup_ZF_1_L4 by auto
  from A3 obtain a where a∈A by auto
  with D1 have T2: L≤a a≤U by auto
  then have T3: L∈G L-1∈ G U∈G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
      group0.inverse_in_group by auto
  let T = RightTranslation(G,P,L)
  let B = RightTranslation(G,P,L-1)(A)
  let g = U·L-1
  have g∈G+
  proof -
    from T2 have L≤U using Group_order_transitive by fast
    with ordGroupAssum T3 have L·L-1≤g
      using IsAnOrdGroup_def by simp
    with T3 show thesis using OrderedGroup_ZF_1_L1 group0.group0_2_L6
      OrderedGroup_ZF_1_L2 by simp
  qed
  moreover from T3 have T : G→G
    using OrderedGroup_ZF_1_L1 group0.group0_5_L1
    by simp
  moreover have A = T(B)
  proof -
    from T3 T1 have T(B) = {a·L-1·L. a∈A}
      using OrderedGroup_ZF_1_L1 group0.group0_5_L6
      by simp
    moreover from T3 T1 have ∀a∈A. a·L-1·L = a·(L-1·L)
      using OrderedGroup_ZF_1_L1 group0.group_oper_assoc by auto
    ultimately have T(B) = {a·(L-1·L). a∈A} by simp
    with T3 have T(B) = {a·1. a∈A}
      using OrderedGroup_ZF_1_L1 group0.group0_2_L6 by simp
    moreover from T1 have ∀a∈A. a·1=a
      using OrderedGroup_ZF_1_L1 group0.group0_2_L2 by auto
    ultimately show thesis by simp
  qed
  moreover have B ⊆ Interval(r,1,g)
  proof
    fix y assume A4: y ∈ B
    def D2: S ≡ RightTranslation(G,P,L-1)
    from D1 have T4: ∀x∈A. L≤x ∧ x≤U by simp
    moreover from D2 have T5: S = RightTranslation(G,P,L-1)
      by simp
```

```
    moreover from A4 D2 have T6: y ∈ S(A) by simp
    ultimately have y≤U·L⁻¹ using OrderedGroup_ZF_2_L1
      by blast
    moreover from T4 T5 T6 have 1≤y by (rule OrderedGroup_ZF_2_L1)
    ultimately show y ∈ Interval(r,1,g) using Interval_def by auto
  qed
  ultimately show
    ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
    by auto
qed
```

If every interval starting at 1 is finite, then every bounded set is finite. I find it interesting that this does not require the group to be linearly ordered (the order to be total).

```
theorem (in group3) OrderedGroup_ZF_2_T1:
  assumes A1: ∀g∈G⁺. Interval(r,1,g) ∈ Fin(G)
  and A2: IsBounded(A,r)
  shows A ∈ Fin(G)
proof -
  from A2 have
    ∃B.∃g∈G⁺.∃T∈G→G. A = T(B) ∧ B ⊆ Interval(r,1,g)
    using OrderedGroup_ZF_2_L2 by simp
  then obtain B g T where D1: g∈G⁺ B ⊆ Interval(r,1,g)
    and D2: T : G→G A = T(B) by auto
  from D1 A1 have B∈Fin(G) using Fin_subset_lemma by blast
  with D2 show thesis using Finite1_L6A by simp
qed
```

In linearly ordered groups finite sets are bounded.

```
theorem (in group3) ord_group_fin_bounded:
  assumes r {is total on} G and B∈Fin(G)
  shows IsBounded(B,r)
  using ordGroupAssum prems IsAnOrdGroup_def IsPartOrder_def Finite_ZF_1_T1
  by simp
```

For nontrivial linearly ordered groups if for every element G we can find one in A that is greater or equal (not necessarily strictly greater), then A can neither be finite nor bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L2A:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  and A3: ∀a∈G. ∃b∈A. a≤b
  shows
  ∀a∈G. ∃b∈A. a≠b ∧ a≤b
  ¬IsBoundedAbove(A,r)
  A ∉ Fin(G)
proof -
  { fix a
    from A1 A2 obtain c where c ∈ G₊
```

```
    using OrderedGroup_ZF_1_L21 by auto
  moreover assume a∈G
  ultimately have
    a·c ∈ G  and I: a < a·c
    using OrderedGroup_ZF_1_L22 by auto
  with A3 obtain b where II: b∈A  and III: a·c ≤ b
    by auto
  moreover from I III have a<b by (rule OrderedGroup_ZF_1_L4A)
  ultimately have ∃b∈A. a≠b ∧ a≤b by auto
} thus ∀a∈G. ∃b∈A. a≠b ∧ a≤b by simp
with ordGroupAssum A1 show
  ¬IsBoundedAbove(A,r)
  A ∉ Fin(G)
  using IsAnOrdGroup_def IsPartOrder_def
  OrderedGroup_ZF_1_L1A Order_ZF_3_L14 Finite_ZF_1_1_L3
  by auto
qed
```

Nontrivial linearly ordered groups are infinite. Recall that Fin(A) is the collection of finite subsets of A. In this lemma we show that $G \notin$ Fin(G), that is that G is not a finite subset of itself. This is a way of saying that G is infinite. We also show that for nontrivial linearly ordered groups G_+ is infinite.

```
theorem (in group3) Linord_group_infinite:
  assumes A1: r {is total on} G and A2: G ≠ {1}
  shows
  G₊ ∉ Fin(G)
  G ∉ Fin(G)
proof -
  from A1 A2 show I: G₊ ∉ Fin(G)
    using OrderedGroup_ZF_1_L23 OrderedGroup_ZF_2_L2A
    by simp
  { assume G ∈ Fin(G)
    moreover have G₊ ⊆ G using PositiveSet_def by auto
    ultimately have G₊ ∈ Fin(G) using Fin_subset_lemma
      by blast
    with I have False by simp
  } then show G ∉ Fin(G) by auto
qed
```

A property of nonempty subsets of linearly ordered groups that don't have a maximum: for any element in such subset we can find one that is strictly greater.

```
lemma (in group3) OrderedGroup_ZF_2_L2B:
  assumes A1: r {is total on} G and A2: A⊆G and
  A3: ¬HasAmaximum(r,A) and A4: x∈A
  shows ∃y∈A. x<y
proof -
```

```
  from ordGroupAssum prems have
    antisym(r)
    r {is total on} G
    A⊆G  ¬HasAmaximum(r,A)  x∈A
    using IsAnOrdGroup_def IsPartOrder_def
    by auto
  then have ∃y∈A. ⟨x,y⟩ ∈ r ∧ y≠x
    using Order_ZF_4_L16 by simp
  then show ∃y∈A. x<y by auto
qed
```

In linearly ordered groups $G \setminus G_+$ is bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L3:
  assumes A1: r {is total on} G shows IsBoundedAbove(G-G+,r)
proof -
  from A1 have ∀a∈G-G+. a≤1
    using OrderedGroup_ZF_1_L17 by simp
  then show IsBoundedAbove(G-G+,r)
    using IsBoundedAbove_def by auto
qed
```

In linearly ordered groups if $A \cap G_+$ is finite, then A is bounded above.

```
lemma (in group3) OrderedGroup_ZF_2_L4:
  assumes A1: r {is total on} G and A2: A⊆G
  and A3: A ∩ G+ ∈ Fin(G)
  shows IsBoundedAbove(A,r)
proof -
  have A ∩ (G-G+) ⊆ G-G+ by auto
  with A1 have IsBoundedAbove(A ∩ (G-G+),r)
    using OrderedGroup_ZF_2_L3 Order_ZF_3_L13
    by blast
  moreover from A1 A3 have IsBoundedAbove(A ∩ G+,r)
    using ord_group_fin_bounded IsBounded_def
    by simp
  moreover from A1 ordGroupAssum have
    r {is total on} G  trans(r)  r⊆G×G
    using IsAnOrdGroup_def IsPartOrder_def by auto
  ultimately have IsBoundedAbove(A ∩ (G-G+) ∪ A ∩ G+,r)
    using Order_ZF_3_L3 by simp
  moreover from A2 have A = A ∩ (G-G+) ∪ A ∩ G+
    by auto
  ultimately show  IsBoundedAbove(A,r) by simp
qed
```

If a set $-A \subseteq G$ is bounded above, then A is bounded below.

```
lemma (in group3) OrderedGroup_ZF_2_L5:
  assumes A1: A⊆G and A2: IsBoundedAbove(-A,r)
  shows IsBoundedBelow(A,r)
proof (cases A = 0)
```

```
  assume A = 0 show IsBoundedBelow(A,r)
    using IsBoundedBelow_def by auto
next assume A3: A≠0
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  with A1 A2 A3 obtain u where D: ∀a∈(-A). a≤u
    using func1_1_L15A IsBoundedAbove_def by auto
  { fix b assume b∈A
    with A1 I D have b⁻¹ ≤ u and T: b∈G
      using func_imagedef by auto
    then have u⁻¹≤(b⁻¹)⁻¹ using OrderedGroup_ZF_1_L5
      by simp
    with T have u⁻¹≤b
      using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp
  } then have ∀b∈A. ⟨u⁻¹,b⟩ ∈ r by simp
  then show IsBoundedBelow(A,r)
    using Order_ZF_3_L9 by blast
qed
```

if $a \leq b$, then the image of the interval $a..b$ by any function is nonempty.

```
lemma (in group3) OrderedGroup_ZF_2_L6:
  assumes a≤b and f:G→G
  shows f(Interval(r,a,b)) ≠ 0
  using ordGroupAssum prems OrderedGroup_ZF_1_L4
    Order_ZF_2_L6 Order_ZF_2_L2A
    IsAnOrdGroup_def IsPartOrder_def func1_1_L15A
  by auto
```

17.4 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:
  AbsoluteValue(G,P,r) : G→G
proof -
  let f = id(G⁺)
  let g = restrict(GroupInv(G,P),G-G⁺)
  have f : G⁺→G⁺ using id_type by simp
  then have f : G⁺→G using OrderedGroup_ZF_1_L4E
    by (rule fun_weaken_type)
  moreover have g : G-G⁺→G
  proof -
    from ordGroupAssum have GroupInv(G,P) : G→G
      using IsAnOrdGroup_def group0_2_T2 by simp
    moreover have G-G⁺ ⊆ G by auto
    ultimately show thesis using restrict_type2 by simp
  qed
```

```
  moreover have G⁺∩(G-G⁺) = 0 by blast
  ultimately have f ∪ g : G⁺∪(G-G⁺)→G∪G
    by (rule fun_disjoint_Un)
  moreover have G⁺∪(G-G⁺) = G using OrderedGroup_ZF_1_L4E
    by auto
  ultimately show AbsoluteValue(G,P,r) : G→G
    using AbsoluteValue_def by simp
qed
```

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

```
lemma (in group3) OrderedGroup_ZF_3_L2:
  assumes A1: a∈G⁺ shows |a| = a
proof -
  from ordGroupAssum have GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  with A1 show thesis using
    func1_1_L1 OrderedGroup_ZF_1_L4E fun_disjoint_apply1
    AbsoluteValue_def id_conv by simp
qed
```

```
lemma (in group3) OrderedGroup_ZF_3_L2A:
  shows |1| = 1 using OrderedGroup_ZF_1_L3A OrderedGroup_ZF_3_L2
  by simp
```

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

```
lemma (in group3) OrderedGroup_ZF_3_L2B:
  assumes a∈G₊ shows |a| = a
  using prems PositiveSet_def Nonnegative_def OrderedGroup_ZF_3_L2
  by auto
```

If $a \in G \setminus G^+$, then $|a| = a^{-1}$.

```
lemma (in group3) OrderedGroup_ZF_3_L3:
   assumes A1: a ∈ G-G⁺ shows |a| = a⁻¹
proof -
  have domain(id(G⁺)) = G⁺
    using id_type func1_1_L1 by auto
  with A1 show thesis using fun_disjoint_apply2 AbsoluteValue_def
    restrict by simp
qed
```

For elements that not greater than the unit, the absolute value is the inverse.

```
lemma (in group3) OrderedGroup_ZF_3_L3A:
  assumes A1: a≤1
  shows |a| = a⁻¹
proof (cases a=1)
  assume a=1 then show |a| = a⁻¹
    using OrderedGroup_ZF_3_L2A OrderedGroup_ZF_1_L1 group0.group_inv_of_one
    by simp
```

```
next assume a≠1
  with A1 show |a| = a⁻¹ using OrderedGroup_ZF_1_L4C OrderedGroup_ZF_3_L3
    by simp
qed
```

In linearly ordered groups the absolute value of any element is in G^+.

```
lemma (in group3) OrderedGroup_ZF_3_L3B:
  assumes A1: r {is total on} G and A2: a∈G
  shows |a| ∈ G+
proof (cases a∈G+)
  assume a ∈ G+ then show |a| ∈ G+
    using OrderedGroup_ZF_3_L2 by simp
next assume a ∉ G+
  with A1 A2 show |a| ∈ G+ using OrderedGroup_ZF_3_L3
    OrderedGroup_ZF_1_L6 by simp
qed
```

For linearly ordered groups (where the order is total), the absolute value maps the group into the positive set.

```
lemma (in group3) OrderedGroup_ZF_3_L3C:
  assumes A1: r {is total on} G
  shows AbsoluteValue(G,P,r) : G→G+
proof-
  have AbsoluteValue(G,P,r) : G→G using OrderedGroup_ZF_3_L1
    by simp
  moreover from A1 have T2:
    ∀g∈G. AbsoluteValue(G,P,r)(g)  ∈ G+
    using OrderedGroup_ZF_3_L3B by simp
  ultimately show thesis by (rule func1_1_L1A)
qed
```

If the absolute value is the unit, then the elemnent is the unit.

```
lemma (in group3) OrderedGroup_ZF_3_L3D:
  assumes A1: a∈G and A2: |a| = 1
  shows a = 1
proof (cases a∈G+)
  assume a ∈ G+
  with A2 show a = 1 using OrderedGroup_ZF_3_L2 by simp
next assume a ∉ G+
  with A1 A2 show a = 1 using
    OrderedGroup_ZF_3_L3 OrderedGroup_ZF_1_L1 group0.group0_2_L8A
    by auto
qed
```

In linearly ordered groups the unit is not greater than the absolute value of any element.

```
lemma (in group3) OrderedGroup_ZF_3_L3E:
  assumes r {is total on} G and a∈G
```

```
  shows 1 ≤ |a|
  using prems OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L2 by simp
```

If b is greater than both a and a^{-1}, then b is greater than $|a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L4:
  assumes A1: a≤b and A2: a⁻¹≤ b
  shows |a|≤ b
proof (cases a∈G⁺)
  assume a∈G⁺
  with A1 show |a|≤ b using OrderedGroup_ZF_3_L2 by simp
next assume a∉G⁺
  with A1 A2 show |a|≤ b
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_3_L3 by simp
qed
```

In linearly ordered groups $a \leq |a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L5:
  assumes A1: r {is total on} G and A2: a∈G
  shows a ≤ |a|
proof (cases a∈G⁺)
  assume a ∈ G⁺
  with A2 show a ≤ |a|
    using OrderedGroup_ZF_3_L2 OrderedGroup_ZF_1_L3 by simp
next assume a ∉ G⁺
  with A1 A2 show a ≤ |a|
    using OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L4B by simp
qed
```

$a^{-1} \leq |a|$ (in additive notation it would be $-a \leq |a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L6:
  assumes A1: a∈G shows a⁻¹ ≤ |a|
proof (cases a∈G⁺)
  assume a ∈ G⁺
  then have T1: 1≤a and T2: |a| = a using OrderedGroup_ZF_1_L2
    OrderedGroup_ZF_3_L2 by auto
  then have a⁻¹≤1⁻¹ using OrderedGroup_ZF_1_L5 by simp
  then have T3: a⁻¹≤1
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_one by simp
  from T3 T1 have a⁻¹≤a by (rule Group_order_transitive)
  with T2 show a⁻¹ ≤ |a| by simp
next assume A2: a ∉ G⁺
  from A1 have |a| ∈ G
    using OrderedGroup_ZF_3_L1 apply_funtype by auto
  with ordGroupAssum have |a| ≤ |a|
    using IsAnOrdGroup_def IsPartOrder_def refl_def by simp
  with A1 A2 show a⁻¹ ≤ |a| using OrderedGroup_ZF_3_L3 by simp
qed
```

Some inequalities about the product of two elements of a linearly ordered group and its absolute value.

```
lemma (in group3) OrderedGroup_ZF_3_L6A:
  assumes r {is total on} G and a∈G  b∈G
  shows
  a·b ≤|a|·|b|
  a·b⁻¹ ≤|a|·|b|
  a⁻¹·b ≤|a|·|b|
  a⁻¹·b⁻¹ ≤|a|·|b|
  using prems OrderedGroup_ZF_3_L5 OrderedGroup_ZF_3_L6
    OrderedGroup_ZF_1_L5B by auto
```

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

```
lemma (in group3) OrderedGroup_ZF_3_L7:
  assumes r {is total on} G and a∈G
  shows |a⁻¹|≤|a|
  using prems OrderedGroup_ZF_3_L5 OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    OrderedGroup_ZF_3_L6 OrderedGroup_ZF_3_L4 by simp
```

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

```
lemma (in group3) OrderedGroup_ZF_3_L7A:
  assumes A1: r {is total on} G and A2: a∈G
  shows |a⁻¹| = |a|
proof -
  from A2 have a⁻¹∈G using OrderedGroup_ZF_1_L1 group0.inverse_in_group
    by simp
  with A1 have |(a⁻¹)⁻¹| ≤ |a⁻¹| using OrderedGroup_ZF_3_L7 by simp
  with A1 A2 have |a⁻¹| ≤ |a|  |a| ≤ |a⁻¹|
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv OrderedGroup_ZF_3_L7
    by auto
  then show thesis by (rule group_order_antisym)
qed
```

$|a \cdot b^{-1}| = |b \cdot a^{-1}|$. It doesn't look so strange in the additive notation: $|a - b| = |b - a|$.

```
lemma (in group3) OrderedGroup_ZF_3_L7B:
  assumes A1: r {is total on} G and A2: a∈G b∈G
  shows |a·b⁻¹| = |b·a⁻¹|
proof -
  from A1 A2 have |(a·b⁻¹)⁻¹| = |a·b⁻¹| using
    OrderedGroup_ZF_1_L1 group0.inverse_in_group group0.group0_2_L1
    monoid0.group0_1_L1 OrderedGroup_ZF_3_L7A by simp
  moreover from A2 have (a·b⁻¹)⁻¹ =  b·a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L12 by simp
  ultimately show thesis by simp
qed
```

Triangle inequality for linearly ordered abelian groups. It would be nice to drop commutativity or give an example that shows we can't do that.

```
theorem (in group3) OrdGroup_triangle_ineq:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G
  shows |a·b| ≤ |a|·|b|
proof -
  from A1 A2 A3 have
    a ≤ |a| b ≤ |b| a⁻¹ ≤ |a| b⁻¹ ≤ |b|
    using OrderedGroup_ZF_3_L5 OrderedGroup_ZF_3_L6 by auto
  then have a·b ≤ |a|·|b| a⁻¹·b⁻¹ ≤ |a|·|b|
    using OrderedGroup_ZF_1_L5B by auto
  with A1 A3 show |a·b| ≤ |a|·|b|
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_two IsCommutative_def

    OrderedGroup_ZF_3_L4 by simp
qed
```

We can multiply the sides of an inequality with absolute value.

```
lemma (in group3) OrderedGroup_ZF_3_L7C:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G b∈G
  and A4: |a| ≤ c  |b| ≤ d
  shows |a·b| ≤ c·d
proof -
  from A1 A2 A3 A4 have |a·b| ≤ |a|·|b|
    using OrderedGroup_ZF_1_L4 OrdGroup_triangle_ineq
    by simp
  moreover from A4 have |a|·|b| ≤ c·d
    using OrderedGroup_ZF_1_L5B by simp
  ultimately show thesis by (rule Group_order_transitive)
qed
```

A version of the `OrderedGroup_ZF_3_L7C` but with multiplying by the inverse.

```
lemma (in group3) OrderedGroup_ZF_3_L7CA:
  assumes P {is commutative on} G
  and r {is total on} G and a∈G  b∈G
  and |a| ≤ c  |b| ≤ d
  shows |a·b⁻¹| ≤ c·d
  using prems OrderedGroup_ZF_1_L1 group0.inverse_in_group
  OrderedGroup_ZF_3_L7A OrderedGroup_ZF_3_L7C by simp
```

Triangle inequality with three integers.

```
lemma (in group3) OrdGroup_triangle_ineq3:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G  c∈G
  shows |a·b·c| ≤ |a|·|b|·|c|
proof -
  from A3 have T: a·b ∈ G  |c| ∈ G
    using OrderedGroup_ZF_1_L1 group0.group_op_closed
      OrderedGroup_ZF_3_L1 apply_funtype by auto
```

```
  with A1 A2 A3 have |a·b·c| ≤ |a·b|·|c|
    using OrdGroup_triangle_ineq by simp
  moreover from ordGroupAssum A1 A2 A3 T have
    |a·b|·|c| ≤ |a|·|b|·|c|
    using OrdGroup_triangle_ineq IsAnOrdGroup_def by simp
  ultimately show |a·b·c| ≤ |a|·|b|·|c|
    by (rule Group_order_transitive)
qed
```

Some variants of the triangle inequality.

```
lemma (in group3) OrderedGroup_ZF_3_L7D:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G
  and A4: |a·b⁻¹| ≤ c
  shows
  |a| ≤ c·|b|
  |a| ≤ |b|·c
  c⁻¹·a ≤ b
  a·c⁻¹ ≤ b
  a ≤ b·c
proof -
  from A3 A4 have
    T: a·b⁻¹ ∈ G  |b| ∈ G  c∈G  c⁻¹ ∈ G
    using OrderedGroup_ZF_1_L1
      group0.inverse_in_group group0.group0_2_L1 monoid0.group0_1_L1
      OrderedGroup_ZF_3_L1 apply_funtype  OrderedGroup_ZF_1_L4
    by auto
  from A3 have |a| = |a·b⁻¹·b|
    using OrderedGroup_ZF_1_L1 group0.group0_2_L16
    by simp
  with A1 A2 A3 T have |a| ≤ |a·b⁻¹|·|b|
    using OrdGroup_triangle_ineq by simp
  with T A4 show |a| ≤ c·|b| using OrderedGroup_ZF_1_L5C
    by blast
  with T A1 show |a| ≤ |b|·c
    using IsCommutative_def by simp
  from A2 T have a·b⁻¹ ≤ |a·b⁻¹|
    using OrderedGroup_ZF_3_L5 by simp
  moreover from A4 have |a·b⁻¹| ≤ c .
  ultimately have I: a·b⁻¹ ≤ c
    by (rule Group_order_transitive)
  with A3 show c⁻¹·a ≤ b
    using OrderedGroup_ZF_1_L5H by simp
  with A1 A3 T show a·c⁻¹ ≤ b
    using IsCommutative_def by simp
  from A1 A3 T I show a ≤ b·c
    using OrderedGroup_ZF_1_L5H IsCommutative_def
    by auto
qed
```

Some more variants of the triangle inequality.

```
lemma (in group3) OrderedGroup_ZF_3_L7E:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and A3: a∈G  b∈G
  and A4: |a·b⁻¹| ≤ c
  shows b·c⁻¹ ≤ a
proof -
  from A3 have a·b⁻¹ ∈ G
    using OrderedGroup_ZF_1_L1
      group0.inverse_in_group group0.group_op_closed
    by auto
  with A2 have |(a·b⁻¹)⁻¹| = |a·b⁻¹|
    using OrderedGroup_ZF_3_L7A by simp
  moreover from A3 have (a·b⁻¹)⁻¹ = b·a⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_2_L12
    by simp
  ultimately have |b·a⁻¹| = |a·b⁻¹|
    by simp
  with A1 A2 A3 A4 show b·c⁻¹ ≤ a
    using OrderedGroup_ZF_3_L7D by simp
qed
```

An application of the triangle inequality with four group elements.

```
lemma (in group3) OrderedGroup_ZF_3_L7F:
  assumes A1: P {is commutative on} G
  and A2: r {is total on} G and
  A3: a∈G  b∈G  c∈G  d∈G
  shows |a·c⁻¹| ≤ |a·b|·|c·d|·|b·d⁻¹|
proof -
  from A3 have T:
    a·c⁻¹ ∈ G  a·b ∈ G  c·d ∈ G  b·d⁻¹ ∈ G
    (c·d)⁻¹ ∈ G  (b·d⁻¹)⁻¹ ∈ G
    using OrderedGroup_ZF_1_L1
      group0.inverse_in_group group0.group_op_closed
    by auto
  with A1 A2 have |(a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹| ≤ |a·b|·|(c·d)⁻¹|·|(b·d⁻¹)⁻¹|
    using OrdGroup_triangle_ineq3 by simp
  moreover from A2 T have |(c·d)⁻¹| =|c·d| and |(b·d⁻¹)⁻¹| = |b·d⁻¹|
    using OrderedGroup_ZF_3_L7A by auto
  moreover from A1 A3 have (a·b)·(c·d)⁻¹·(b·d⁻¹)⁻¹ = a·c⁻¹
    using OrderedGroup_ZF_1_L1 group0.group0_4_L8
    by simp
  ultimately show |a·c⁻¹| ≤ |a·b|·|c·d|·|b·d⁻¹|
    by simp
qed
```

$|a| \leq L$ implies $L^{-1} \leq a$ (it would be $-L \leq a$ in the additive notation).

```
lemma (in group3) OrderedGroup_ZF_3_L8:
  assumes A1:  a∈G and A2: |a|≤L
```

```
  shows
 L⁻¹≤a
proof -
 from A1 have I: a⁻¹ ≤ |a| using OrderedGroup_ZF_3_L6 by simp
 from I A2 have a⁻¹ ≤ L by (rule Group_order_transitive)
 then have L⁻¹≤(a⁻¹)⁻¹ using OrderedGroup_ZF_1_L5 by simp
 with A1 show L⁻¹≤a using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
   by simp
qed
```

In linearly ordered groups $|a| \leq L$ implies $a \leq L$ (it would be $a \leq L$ in the additive notation).

```
lemma (in group3) OrderedGroup_ZF_3_L8A:
 assumes A1: r {is total on} G
 and A2: a∈G and A3: |a|≤L
 shows
 a≤L
 1≤L
proof -
 from A1 A2 have I: a ≤ |a| using OrderedGroup_ZF_3_L5 by simp
 from I A3 show a≤L by (rule Group_order_transitive)
 from A1 A2 A3 have 1 ≤ |a|  |a|≤L
    using OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L2 by auto
  then show 1≤L by (rule Group_order_transitive)
qed
```

A somewhat generalized version of the above lemma.

```
lemma (in group3) OrderedGroup_ZF_3_L8B:
 assumes A1: a∈G and A2: |a|≤L and A3: 1≤c
 shows (L·c)⁻¹ ≤ a
proof -
 from A1 A2 A3 have c⁻¹·L⁻¹ ≤ 1·a
   using OrderedGroup_ZF_3_L8 OrderedGroup_ZF_1_L5AB
   OrderedGroup_ZF_1_L5B by simp
 with A1 A2 A3 show (L·c)⁻¹ ≤ a
   using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_1_L1
     group0.group_inv_of_two group0.group0_2_L2
   by simp
qed
```

If b is between a and $a \cdot c$, then $b \cdot a^{-1} \leq c$.

```
lemma (in group3) OrderedGroup_ZF_3_L8C:
 assumes A1: a≤b and A2: c∈G and A3: b≤c·a
 shows |b·a⁻¹| ≤ c
proof -
 from A1 A2 A3 have b·a⁻¹ ≤ c
   using OrderedGroup_ZF_1_L9C OrderedGroup_ZF_1_L4
   by simp
 moreover have (b·a⁻¹)⁻¹ ≤ c
```

```
  proof -
    from A1 have T: a∈G  b∈G
      using OrderedGroup_ZF_1_L4 by auto
    with A1 have a·b⁻¹ ≤ 1
      using OrderedGroup_ZF_1_L9 by blast
    moreover
    from A1 A3 have a≤c·a
      by (rule Group_order_transitive)
    with ordGroupAssum T have a·a⁻¹ ≤ c·a·a⁻¹
      using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      IsAnOrdGroup_def by simp
    with T A2 have 1 ≤ c
      using OrderedGroup_ZF_1_L1
        group0.group0_2_L6 group0.group0_2_L16
      by simp
    ultimately have a·b⁻¹ ≤ c
      by (rule Group_order_transitive)
    with T show (b·a⁻¹)⁻¹ ≤ c
      using OrderedGroup_ZF_1_L1 group0.group0_2_L12
      by simp
  qed
  ultimately show |b·a⁻¹| ≤ c
    using OrderedGroup_ZF_3_L4 by simp
qed
```

For linearly ordered groups if the absolute values of elements in a set are bounded, then the set is bounded.

```
lemma (in group3) OrderedGroup_ZF_3_L9:
  assumes A1: r {is total on} G
  and A2: A⊆G and A3: ∀a∈A. |a| ≤ L
  shows IsBounded(A,r)
proof -
  from A1 A2 A3 have
    ∀a∈A. a≤L  ∀a∈A. L⁻¹≤a
    using OrderedGroup_ZF_3_L8 OrderedGroup_ZF_3_L8A by auto
  then show IsBounded(A,r) using
    IsBoundedAbove_def IsBoundedBelow_def IsBounded_def
    by auto
qed
```

A slightly more general version of the previous lemma, stating the same fact for a set defined by separation.

```
lemma (in group3) OrderedGroup_ZF_3_L9A:
  assumes A1: r {is total on} G
  and A2: ∀x∈X. b(x)∈G ∧ |b(x)|≤L
  shows IsBounded({b(x). x∈X},r)
proof -
  from A2 have {b(x). x∈X} ⊆ G ∀a∈{b(x). x∈X}. |a| ≤ L
    by auto
```

```
  with A1 show thesis using OrderedGroup_ZF_3_L9 by blast
qed
```

A special form of the previous lemma stating a similar fact for an image of a set by a function with values in a linearly ordered group.

```
lemma (in group3) OrderedGroup_ZF_3_L9B:
  assumes A1: r {is total on} G
  and A2: f:X→G and A3: A⊆X
  and A4: ∀x∈A. |f(x)| ≤ L
  shows IsBounded(f(A),r)
proof -
  from A2 A3 A4 have ∀x∈A. f(x) ∈ G ∧  |f(x)| ≤ L
    using apply_funtype by auto
  with A1 have  IsBounded({f(x). x∈A},r)
    by (rule OrderedGroup_ZF_3_L9A)
  with A2 A3 show IsBounded(f(A),r)
    using func_imagedef by simp
qed
```

For linearly ordered groups if $l \leq a \leq u$ then $|a|$ is smaller than the greater of $|l|, |u|$.

```
lemma (in group3) OrderedGroup_ZF_3_L10:
  assumes A1: r {is total on} G
  and A2: l≤a  a≤u
  shows
  |a| ≤ GreaterOf(r,|l|,|u|)
proof (cases a∈G+)
  from A2 have T1: |l| ∈ G |a| ∈ G |u| ∈ G
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_3_L1 apply_funtype
    by auto
  assume A3: a∈G+
  with A2 have 1≤a a≤u
    using OrderedGroup_ZF_1_L2 by auto
  then have 1≤u by (rule Group_order_transitive)
  with A2 A3 have |a|≤|u|
    using OrderedGroup_ZF_1_L2 OrderedGroup_ZF_3_L2 by simp
  moreover from A1 T1 have |u| ≤ GreaterOf(r,|l|,|u|)
    using Order_ZF_3_L2 by simp
  ultimately show |a| ≤ GreaterOf(r,|l|,|u|)
    by (rule Group_order_transitive)
next assume A4: a∉G+
  with A2 have T2:
    l∈G |l| ∈ G |a| ∈ G |u| ∈ G a ∈ G-G+
    using OrderedGroup_ZF_1_L4 OrderedGroup_ZF_3_L1 apply_funtype
    by auto
  with A2 have l ∈ G-G+ using OrderedGroup_ZF_1_L4D by fast
  with T2 A2 have |a| ≤ |l|
    using OrderedGroup_ZF_3_L3 OrderedGroup_ZF_1_L5
    by simp
```

```
  moreover from A1 T2 have |l| ≤ GreaterOf(r,|l|,|u|)
    using Order_ZF_3_L2 by simp
  ultimately show |a| ≤ GreaterOf(r,|l|,|u|)
    by (rule Group_order_transitive)
qed
```

For linearly ordered groups if a set is bounded then the absolute values are bounded.

```
lemma (in group3) OrderedGroup_ZF_3_L10A:
  assumes A1: r {is total on} G
  and A2: IsBounded(A,r)
  shows ∃L. ∀a∈A. |a| ≤ L
proof (cases A=0)
  assume A = 0 then show thesis by auto
next assume A3: A≠0
  with A2 obtain u l where ∀g∈A. l≤g ∧  g≤u
    using IsBounded_def IsBoundedAbove_def IsBoundedBelow_def
    by auto
  with A1 have ∀a∈A. |a| ≤ GreaterOf(r,|l|,|u|)
    using OrderedGroup_ZF_3_L10 by simp
  then show thesis by auto
qed
```

A slightly more general version of the previous lemma, stating the same fact for a set defined by separation.

```
lemma (in group3) OrderedGroup_ZF_3_L11:
  assumes A1: r {is total on} G
  and A2: IsBounded({b(x).x∈X},r)
  shows ∃L. ∀x∈X. |b(x)| ≤ L
proof -
  from A1 A2 show thesis using OrderedGroup_ZF_3_L10A
    by blast
qed
```

Absolute values of elements of a finite image of a nonempty set are bounded by an element of the group.

```
lemma (in group3) OrderedGroup_ZF_3_L11A:
  assumes A1: r {is total on} G
  and A2: X≠0 and A3: {b(x). x∈X} ∈ Fin(G)
  shows ∃L∈G. ∀x∈X. |b(x)| ≤ L
proof -
  from A1 A3 have ∃L. ∀x∈X. |b(x)| ≤ L
     using  ord_group_fin_bounded OrderedGroup_ZF_3_L11
     by simp
  then obtain L where I: ∀x∈X. |b(x)| ≤ L
    using OrderedGroup_ZF_3_L11 by auto
  from A2 obtain x where x∈X by auto
  with I show thesis using OrderedGroup_ZF_1_L4
```

```
    by blast
qed
```

In totally oredered groups the absolute value of a nonunit element is in G_+.

```
lemma (in group3) OrderedGroup_ZF_3_L12:
  assumes A1: r {is total on} G
  and A2: a∈G  and A3: a≠1
  shows |a| ∈ G+
proof -
  from A1 A2 have |a| ∈ G  1 ≤ |a|
    using OrderedGroup_ZF_3_L1 apply_funtype
      OrderedGroup_ZF_3_L3B OrderedGroup_ZF_1_L2
    by auto
  moreover from A2 A3 have |a| ≠ 1
    using OrderedGroup_ZF_3_L3D by auto
  ultimately show |a| ∈ G+
    using PositiveSet_def by auto
qed
```

17.5 Maximum absolute value of a set

Quite often when considering inequalities we prefer to talk about the absolute values instead of raw elements of a set. This section formalizes some material that is useful for that.

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum belongs to the image of the set by the absolute value function.

```
lemma (in group3) OrderedGroup_ZF_4_L1:
  assumes A ⊆ G
  and HasAmaximum(r,A) HasAminimum(r,A)
  and M = GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
  shows M ∈ AbsoluteValue(G,P,r)(A)
  using ordGroupAssum prems IsAnOrdGroup_def IsPartOrder_def
    Order_ZF_4_L3 Order_ZF_4_L4 OrderedGroup_ZF_3_L1
    func_imagedef GreaterOf_def by auto
```

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum bounds absolute values of all elements of the set.

```
lemma (in group3) OrderedGroup_ZF_4_L2:
  assumes A1: r {is total on} G
  and A2: HasAmaximum(r,A)  HasAminimum(r,A)
  and A3: a∈A
  shows |a|≤ GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
proof -
  from ordGroupAssum A2 A3 have
    Minimum(r,A)≤ a a≤ Maximum(r,A)
```

```
    using IsAnOrdGroup_def IsPartOrder_def Order_ZF_4_L3 Order_ZF_4_L4
    by auto
  with A1 show thesis by (rule OrderedGroup_ZF_3_L10)
qed
```

If a set has a maximum and minimum, then the greater of the absolute value of the maximum and minimum bounds absolute values of all elements of the set. In this lemma the absolute values of ekements of a set are represented as the elements of the image of the set by the absolute value function.

```
lemma (in group3) OrderedGroup_ZF_4_L3:
  assumes r {is total on} G and A ⊆ G
  and HasAmaximum(r,A) HasAminimum(r,A)
  and b ∈ AbsoluteValue(G,P,r)(A)
  shows b≤ GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
  using prems OrderedGroup_ZF_3_L1 func_imagedef OrderedGroup_ZF_4_L2
  by auto
```

If a set has a maximum and minimum, then the set of absolute values also has a maximum.

```
lemma (in group3) OrderedGroup_ZF_4_L4:
  assumes A1: r {is total on} G and A2: A ⊆ G
  and A3: HasAmaximum(r,A) HasAminimum(r,A)
  shows HasAmaximum(r,AbsoluteValue(G,P,r)(A))
proof -
  let M = GreaterOf(r,|Minimum(r,A)|,|Maximum(r,A)|)
  from A2 A3 have M ∈ AbsoluteValue(G,P,r)(A)
    using OrderedGroup_ZF_4_L1 by simp
  moreover from A1 A2 A3 have
    ∀b ∈ AbsoluteValue(G,P,r)(A). b ≤ M
    using OrderedGroup_ZF_4_L3 by simp
  ultimately show thesis using HasAmaximum_def by auto
qed
```

If a set has a maximum and a minimum, then all absolute values are bounded by the maximum of the set of absolute values.

```
lemma (in group3) OrderedGroup_ZF_4_L5:
  assumes A1: r {is total on} G and A2: A ⊆ G
  and A3: HasAmaximum(r,A) HasAminimum(r,A)
  and A4: a∈A
  shows |a| ≤ Maximum(r,AbsoluteValue(G,P,r)(A))
proof -
  from A2 A4 have |a| ∈ AbsoluteValue(G,P,r)(A)
    using OrderedGroup_ZF_3_L1 func_imagedef by auto
  with ordGroupAssum A1 A2 A3 show thesis using
    IsAnOrdGroup_def IsPartOrder_def OrderedGroup_ZF_4_L4
    Order_ZF_4_L3 by simp
qed
```

17.6 Alternative definitions

Sometimes it is usful to define the order by prescibing the set of positive or nonnegative elements. This section deals with two such definitions. One takes a subset H of G that is closed under the group operation, $1 \notin H$ and for every $a \in H$ we have either $a \in H$ or $a^{-1} \in H$. Then the order is defined as $a \leq b$ iff $a = b$ or $a^{-1}b \in H$. For abelian groups this makes a linearly ordered group. We will refer to order defined this way in the comments as the order defined by a positive set. The context used in this section is the `group0` context defined in `Group_ZF` theory. Recall that `f` in that context denotes the group operation (unlike in the previous sections where the group operation was denoted `P`.

The order defined by a positive set is the same as the order defined by a nonnegative set.

```
lemma (in group0) OrderedGroup_ZF_5_L1:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  shows ⟨a,b⟩ ∈ r  ⟷ a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
proof
  assume ⟨a,b⟩ ∈ r
  with A1 show a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
    using group0_2_L6 by auto
next assume a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
   then have a∈G ∧ b∈G ∧ b=(a⁻¹)⁻¹ ∨  a∈G ∧ b∈G ∧ a⁻¹·b ∈ H
    using  inverse_in_group group0_2_L9 by auto
  with A1 show ⟨a,b⟩ ∈ r using group_inv_of_inv
    by auto
qed
```

The relation defined by a positive set is antisymmetric.

```
lemma (in group0) OrderedGroup_ZF_5_L2:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  and A2: ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  shows antisym(r)
proof -
  { fix a b assume A3: ⟨a,b⟩ ∈ r  ⟨b,a⟩ ∈ r
    with A1 have T: a∈G  b∈G by auto
    { assume A4: a≠b
      with A1 A3 have a⁻¹·b ∈ G  a⁻¹·b ∈ H  (a⁻¹·b)⁻¹ ∈ H
        using inverse_in_group group0_2_L1 monoid0.group0_1_L1 group0_2_L12
        by auto
      with A2 have a⁻¹·b = 1 using Xor_def by auto
      with T A4 have False using group0_2_L11 by auto
    } then have a=b by auto
  } then show antisym(r) by (rule antisymI)
qed
```

The relation defined by a positive set is transitive.

```
lemma (in group0) OrderedGroup_ZF_5_L3:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  and A2: H⊆G  H {is closed under} f
  shows trans(r)
proof -
  { fix a b c assume ⟨a,b⟩ ∈ r  ⟨b,c⟩ ∈ r
    with A1 have
      a∈G ∧ b∈G ∧ a⁻¹·b ∈ H ∪ {1}
      b∈G ∧ c∈G ∧ b⁻¹·c ∈ H ∪ {1}
      using OrderedGroup_ZF_5_L1 by auto
    with A2 have
      I: a∈G  b∈G  c∈G
      and (a⁻¹·b)·(b⁻¹·c) ∈  H ∪ {1}
      using inverse_in_group group0_2_L17 IsOpClosed_def
      by auto
    moreover from I have a⁻¹·c = (a⁻¹·b)·(b⁻¹·c)
      by (rule group0_2_L14A)
    ultimately have ⟨a,c⟩ ∈ G×G  a⁻¹·c  ∈  H ∪ {1}
      by auto
    with A1 have ⟨a,c⟩ ∈ r using OrderedGroup_ZF_5_L1
      by auto
  } then have ∀ a b c. ⟨a, b⟩ ∈ r ∧ ⟨b, c⟩ ∈ r ⟶ ⟨a, c⟩ ∈ r
    by blast
  then show  trans(r) by (rule Fol1_L2)
qed
```

The relation defined by a positive set is translation invariant. With our definition this step requires the group to be abelian.

```
lemma (in group0) OrderedGroup_ZF_5_L4:
  assumes A1: r = {p ∈ G×G. fst(p) = snd(p) ∨ fst(p)⁻¹·snd(p) ∈ H}
  and A2: f {is commutative on} G
  and A3: ⟨a,b⟩ ∈ r  and A4: c∈G
  shows ⟨a·c,b·c⟩ ∈ r ∧ ⟨c·a,c·b⟩ ∈ r
proof
  from A1 A3 A4 have
    I: a∈G  b∈G  a·c ∈ G  b·c ∈ G
    and II: a⁻¹·b ∈ H ∪ {1}
    using OrderedGroup_ZF_5_L1 group_op_closed
    by auto
  with A2 A4 have (a·c)⁻¹·(b·c) ∈ H ∪ {1}
    using group0_4_L6D by simp
  with A1 I show ⟨a·c,b·c⟩ ∈ r using  OrderedGroup_ZF_5_L1
    by auto
  with A2 A4 I show ⟨c·a,c·b⟩ ∈ r
    using IsCommutative_def by simp
qed
```

If $H \subseteq G$ is closed under the group operation $1 \notin H$ and for every $a \in H$ we have either $a \in H$ or $a^{-1} \in H$, then the relation "$\leq$" defined by $a \leq b \Leftrightarrow$

$a^{-1}b \in H$ orders the group G. In such order H may be the set of positive or nonnegative elements.

```
lemma (in group0) OrderedGroup_ZF_5_L5:
  assumes A1: f {is commutative on} G
  and A2: H⊆G  H {is closed under} f
  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,f,r)
  r {is total on} G
  Nonnegative(G,f,r) = PositiveSet(G,f,r) ∪ {1}
proof -
  from groupAssum A2 A3 A4 have
    IsAgroup(G,f)  r⊆G×G  IsPartOrder(G,r)
    using refl_def OrderedGroup_ZF_5_L2 OrderedGroup_ZF_5_L3
      IsPartOrder_def by auto
  moreover from A1 A4 have
    ∀g∈G. ∀a b. <a,b> ∈ r ⟶ ⟨a·g,b·g⟩ ∈ r ∧ ⟨g·a,g·b⟩ ∈ r
    using OrderedGroup_ZF_5_L4 by blast
  ultimately show IsAnOrdGroup(G,f,r)
    using IsAnOrdGroup_def by simp
  then show Nonnegative(G,f,r) = PositiveSet(G,f,r) ∪ {1}
    using group3_def group3.OrderedGroup_ZF_1_L24
    by simp
  { fix a b
    assume T: a∈G  b∈G
    then have T1: a⁻¹·b ∈ G
      using inverse_in_group group_op_closed by simp
    { assume <a,b> ∉ r
      with A4 T have I: a≠b and II: a⁻¹·b ∉ H
        by auto
      from A3 T T1 I have (a⁻¹·b ∈ H) Xor ((a⁻¹·b)⁻¹ ∈ H)
        using group0_2_L11 by auto
      with A4 T II have <b,a> ∈ r
        using Xor_def group0_2_L12 by simp
    } then have <a,b> ∈ r ∨ <b,a> ∈ r by auto
  } then show r {is total on} G using IsTotal_def
    by simp
qed
```

If the set defined as in `OrderedGroup_ZF_5_L4` does not contain the neutral element, then it is the positive set for the resulting order.

```
lemma (in group0) OrderedGroup_ZF_5_L6:
  assumes f {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,f,r) = H
  using prems group_inv_of_one group0_2_L2 PositiveSet_def
  by auto
```

The next definition describes how we construct an order relation from the prescribed set of positive elements.

constdefs
```
  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 simmarize, 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 f {is commutative on} G
  and H⊆G   H {is closed under} f   1 ∉ H
  and ∀a∈G. a≠1 ⟶ (a∈H) Xor (a⁻¹∈H)
  shows
  IsAnOrdGroup(G,f,OrderFromPosSet(G,f,H))
  OrderFromPosSet(G,f,H) {is total on} G
  PositiveSet(G,f,OrderFromPosSet(G,f,H)) = H
  Nonnegative(G,f,OrderFromPosSet(G,f,H)) = H ∪ {1}
  using prems OrderFromPosSet_def OrderedGroup_ZF_5_L5 OrderedGroup_ZF_5_L6
  by auto
```

17.7 Odd Extensions

In this section we verify properties of odd extensions of functions defined on G_+. An odd extension of a function $f : G_+ \to G$ is a function $f^\circ : G \to G$ defined by $f^\circ(x) = f(x)$ if $x \in G_+$, $f(1) = 1$ and $f^\circ(x) = (f(x^{-1}))^{-1}$ for $x < 1$. Such function is the unique odd function that is equal to f when restricted to G_+.

The next lemma is just to see the definition of the odd extension in the notation used in the `group1` context.

```
lemma (in group3) OrderedGroup_ZF_6_L1:
  shows f° = f ∪ {⟨a, (f(a⁻¹))⁻¹⟩. a ∈ -G₊} ∪ {⟨1,1⟩}
  using OddExtension_def by simp
```

A technical lemma that states that from a function defined on `G₊` with values in G we have $(f(a^{-1}))^{-1} \in G$.

```
lemma (in group3) OrderedGroup_ZF_6_L2:
  assumes f: G₊→G and a∈-G₊
  shows
  f(a⁻¹) ∈ G
```

```
  (f(a⁻¹))⁻¹ ∈ G
  using prems OrderedGroup_ZF_1_L27 apply_funtype
    OrderedGroup_ZF_1_L1 group0.inverse_in_group
  by auto
```

The main theorem about odd extensions. It basically says that the odd extension of a function is what we want to to be.

```
lemma (in group3) odd_ext_props:
  assumes A1: r {is total on} G and A2: f: G₊→G
  shows
  f° : G → G
  ∀a∈G₊. (f°)(a) = f(a)
  ∀a∈(-G₊). (f°)(a) = (f(a⁻¹))⁻¹
  (f°)(1) = 1
proof -
  from A1 A2 have I:
    f: G₊→G
    ∀a∈-G₊. (f(a⁻¹))⁻¹ ∈ G
    G₊∩(-G₊) = 0
    1 ∉ G₊∪(-G₊)
    f° = f ∪ {⟨a, (f(a⁻¹))⁻¹⟩. a ∈ -G₊} ∪ {⟨1,1⟩}
    using OrderedGroup_ZF_6_L2 OrdGroup_decomp2 OrderedGroup_ZF_6_L1
    by auto
  then have f°: G₊ ∪ (-G₊) ∪ {1} →G∪G∪{1}
    by (rule func1_1_L11E)
  moreover from A1 have
    G₊ ∪ (-G₊) ∪ {1} = G
    G∪G∪{1} = G
    using OrdGroup_decomp2 OrderedGroup_ZF_1_L1 group0.group0_2_L2
    by auto
  ultimately show f° : G → G by simp
  from I show ∀a∈G₊. (f°)(a) = f(a)
    by (rule func1_1_L11E)
  from I show ∀a∈(-G₊). (f°)(a) = (f(a⁻¹))⁻¹
    by (rule func1_1_L11E)
  from I show (f°)(1) = 1
    by (rule func1_1_L11E)
qed
```

Odd extensions are odd, of course.

```
lemma (in group3) oddext_is_odd:
  assumes A1: r {is total on} G and A2: f: G₊→G
  and A3: a∈G
  shows (f°)(a⁻¹) = ((f°)(a))⁻¹
proof -
  from A1 A3 have a∈G₊ ∨ a ∈ (-G₊) ∨ a=1
    using OrdGroup_decomp2 by blast
  moreover
  { assume a∈G₊
```

```
    with A1 A2 have a⁻¹ ∈ -G₊ and  (f°)(a) = f(a)
      using OrderedGroup_ZF_1_L25 odd_ext_props by auto
    with A1 A2 have
      (f°)(a⁻¹) = (f((a⁻¹)⁻¹))⁻¹  and (f(a))⁻¹ = ((f°)(a))⁻¹
      using odd_ext_props by auto
    with A3 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp }
  moreover
  { assume A4: a ∈ -G₊
    with A1 A2  have a⁻¹ ∈ G₊ and  (f°)(a) = (f(a⁻¹))⁻¹
      using OrderedGroup_ZF_1_L27 odd_ext_props
      by auto
    with A1 A2 A4 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using odd_ext_props OrderedGroup_ZF_6_L2
        OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
      by simp }
  moreover
  { assume a = 1
    with A1 A2 have (f°)(a⁻¹) = ((f°)(a))⁻¹
      using OrderedGroup_ZF_1_L1 group0.group_inv_of_one
        odd_ext_props by simp
  }
  ultimately show (f°)(a⁻¹) = ((f°)(a))⁻¹
    by auto
qed
```

Another way of saying that odd extensions are odd.

```
lemma (in group3) oddext_is_odd_alt:
  assumes A1: r {is total on} G and A2: f: G₊→G
  and A3: a∈G
  shows ((f°)(a⁻¹))⁻¹ = (f°)(a)
proof -
  from A1 A2 have
    f° : G → G
    ∀a∈G. (f°)(a⁻¹) = ((f°)(a))⁻¹
    using odd_ext_props oddext_is_odd by auto
  then have ∀a∈G. ((f°)(a⁻¹))⁻¹ = (f°)(a)
    using OrderedGroup_ZF_1_L1 group0.group0_6_L2 by simp
  with A3 show ((f°)(a⁻¹))⁻¹ = (f°)(a) by simp
qed
```

17.8 Functions with infinite limits

In this section we consider functions $f : G \to G$ with the property that for $f(x)$ is arbitrarily large for large enough x. More precisely, for every $a \in G$ there exist $b \in G_+$ such that for every $x \geq b$ we have $f(x) \geq a$. In a sense this means that $\lim_{x\to\infty} f(x) = \infty$, hence the title of this section. We also prove dual statements for functions such that $\lim_{x\to-\infty} f(x) = -\infty$.

If an image of a set by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in group3) OrderedGroup_ZF_7_L1:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: f:G→G and
  A4: ∀a∈G.∃b∈G+.∀x. b≤x ⟶ a ≤ f(x) and
  A5: A⊆G and
  A6: IsBoundedAbove(f(A),r)
  shows IsBoundedAbove(A,r)
proof -
  { assume ¬IsBoundedAbove(A,r)
    then have I: ∀u. ∃x∈A. ¬(x≤u)
      using IsBoundedAbove_def by auto
    have ∀a∈G. ∃y∈f(A). a≤y
    proof -
      { fix a assume a∈G
        with A4 obtain b where
          II: b∈G+ and III: ∀x. b≤x ⟶ a ≤ f(x)
          by auto
        from I obtain x where IV: x∈A and ¬(x≤b)
          by auto
        with A1 A5 II have
          r {is total on} G
          x∈G  b∈G  ¬(x≤b)
          using PositiveSet_def by auto
        with III have a ≤ f(x)
          using OrderedGroup_ZF_1_L8 by blast
        with A3 A5 IV have ∃y∈f(A). a≤y
          using func_imagedef by auto
      } thus thesis by simp
    qed
    with A1 A2 A6 have False using OrderedGroup_ZF_2_L2A
      by simp
  } thus thesis by auto
qed
```

If an image of a set defined by separation by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in group3) OrderedGroup_ZF_7_L2:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G+.∀y. b≤y ⟶ a ≤ f(y) and
  A6: ∀x∈X. b(x) ∈ G ∧ f(b(x)) ≤ U
  shows ∃u.∀x∈X. b(x) ≤ u
proof -
  let A = {b(x). x∈X}
  from A6 have I: A⊆G by auto
  moreover note prems
  moreover have IsBoundedAbove(f(A),r)
```

```
  proof -
    from A4 A6 I have ∀z∈f(A). ⟨z,U⟩ ∈ r
      using func_imagedef by simp
    then show IsBoundedAbove(f(A),r)
      by (rule Order_ZF_3_L10)
  qed
  ultimately have IsBoundedAbove(A,r) using OrderedGroup_ZF_7_L1
    by simp
  with A3 have ∃u.∀y∈A. y ≤ u
    using IsBoundedAbove_def by simp
  then show ∃u.∀x∈X. b(x) ≤ u by auto
qed
```

If the image of a set defined by separation by a function with infinite negative limit is bounded below, then the set itself is bounded above. This is dual to OrderedGroup_ZF_7_L2.

```
lemma (in group3) OrderedGroup_ZF_7_L3:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ f(y⁻¹) ≤ a and
  A6: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x))
  shows ∃l.∀x∈X. l ≤ b(x)
proof -
  let g = GroupInv(G,P) O f O GroupInv(G,P)
  from ordGroupAssum have I: GroupInv(G,P) : G→G
    using IsAnOrdGroup_def group0_2_T2 by simp
  with A4 have II: ∀x∈G. g(x) = (f(x⁻¹))⁻¹
    using func1_1_L18 by simp
  note A1 A2 A3
  moreover from A4 I have g : G→G
    using comp_fun by blast
  moreover have ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ a ≤ g(y)
  proof -
  { fix a assume A7: a∈G
    then have a⁻¹ ∈ G
      using OrderedGroup_ZF_1_L1 group0.inverse_in_group
      by simp
    with A5 obtain b where
      III: b∈G₊ and ∀y. b≤y ⟶ f(y⁻¹) ≤ a⁻¹
      by auto
    with II A7 have ∀y. b≤y ⟶ a ≤ g(y)
      using OrderedGroup_ZF_1_L5AD OrderedGroup_ZF_1_L4
      by simp
    with III have ∃b∈G₊.∀y. b≤y ⟶ a ≤ g(y)
      by auto
  } then show ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ a ≤ g(y)
    by simp
  qed
  moreover have ∀x∈X. b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
```

```
  proof-
    { fix x assume x∈X
      with A6 have
        T: b(x) ∈ G  b(x)⁻¹ ∈ G and L ≤ f(b(x))
        using OrderedGroup_ZF_1_L1 group0.inverse_in_group
        by auto
      then have (f(b(x)))⁻¹ ≤ L⁻¹
        using OrderedGroup_ZF_1_L5 by simp
      moreover from II T have (f(b(x)))⁻¹ =  g(b(x)⁻¹)
        using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
        by simp
      ultimately have g(b(x)⁻¹) ≤ L⁻¹ by simp
      with T have b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
        by simp
    } then show ∀x∈X. b(x)⁻¹ ∈ G ∧ g(b(x)⁻¹) ≤ L⁻¹
      by simp
  qed
  ultimately have ∃u.∀x∈X. (b(x))⁻¹ ≤ u
    by (rule OrderedGroup_ZF_7_L2)
  then have ∃u.∀x∈X. u⁻¹ ≤ (b(x)⁻¹)⁻¹
    using OrderedGroup_ZF_1_L5 by auto
  with A6 show ∃l.∀x∈X. l ≤ b(x)
    using OrderedGroup_ZF_1_L1 group0.group_inv_of_inv
    by auto
qed
```

The next lemma combines OrderedGroup_ZF_7_L2 and OrderedGroup_ZF_7_L3 to show that if an image of a set defined by separation by a function with infinite limits is bounded, then the set itself i bounded.

```
lemma (in group3) OrderedGroup_ZF_7_L4:
  assumes A1: r {is total on} G and A2: G ≠ {1} and
  A3: X≠0 and A4: f:G→G and
  A5: ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ a ≤ f(y) and
  A6: ∀a∈G.∃b∈G₊.∀y. b≤y ⟶ f(y⁻¹) ≤ a and
  A7: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x)) ∧ f(b(x)) ≤ U
shows ∃M.∀x∈X. |b(x)| ≤ M
proof -
  from A7 have
    I: ∀x∈X. b(x) ∈ G ∧ f(b(x)) ≤ U and
    II: ∀x∈X. b(x) ∈ G ∧ L ≤ f(b(x))
    by auto
  from A1 A2 A3 A4 A5 I have ∃u.∀x∈X. b(x) ≤ u
    by (rule OrderedGroup_ZF_7_L2)
  moreover from  A1 A2 A3 A4 A6 II have ∃l.∀x∈X. l ≤ b(x)
    by (rule OrderedGroup_ZF_7_L3)
  ultimately have ∃u l. ∀x∈X. l≤b(x) ∧ b(x) ≤ u
    by auto
  with A1 have ∃u l.∀x∈X. |b(x)| ≤ GreaterOf(r,|l|,|u|)
    using OrderedGroup_ZF_3_L10 by blast
```

```
  then show ∃M.∀x∈X. |b(x)| ≤ M
    by auto
qed


end
```

18 Ring_ZF.thy

theory Ring_ZF **imports** Group_ZF

begin

This theory file covers basic facts about rings.

18.1 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. We also define the notion of having no zero divisors.

```
constdefs
  IsAring(R,A,M) ≡ IsAgroup(R,A) ∧ (A {is commutative on} R) ∧
  IsAmonoid(R,M) ∧ IsDistributive(R,A,M)

  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]: a+b ≡ A<a,b>

  fixes ringminus (- _ 89)
  defines ringminus_def [simp]: (-a) ≡ GroupInv(R,A)(a)

  fixes ringsub (infixl - 90)
  defines ringsub_def [simp]: a-b ≡ a+(-b)

  fixes ringm (infixl · 95)
  defines ringm_def [simp]: a·b ≡ M<a,b>

  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]: a² ≡ a·a
```

In the ring0 context we can use theorems proven in some other contexts.

```
lemma (in ring0) Ring_ZF_1_L1: shows
  monoid0(R,M)
  group0(R,A)
  A {is commutative on} R
  using ringAssum IsAring_def group0_def monoid0_def by auto
```

The additive operation in a ring is distributive with respect to the multiplicative operation.

```
lemma (in ring0) ring_oper_distr: assumes A1: a∈R  b∈R  c∈R
  shows
  a·(b+c) = a·b + a·c
  (b+c)·a = b·a + c·a
  using ringAssum prems IsAring_def IsDistributive_def by auto
```

Zero and one of the ring are elements of the ring. The negative of zero is zero.

```
lemma (in ring0) Ring_ZF_1_L2:
  shows 0∈R  1∈R   (-0) = 0
  using Ring_ZF_1_L1 group0.group0_2_L2 monoid0.group0_1_L3
    group0.group_inv_of_one by auto
```

The next lemma lists some properties of a ring that require one element of a ring.

```
lemma (in ring0) Ring_ZF_1_L3: assumes a∈R
  shows
  (-a) ∈ R
  (-(-a)) = a
  a+0 = a
  0+a = a
  a·1 = a
  1·a = a
  a-a = 0
  a-0 = a
  2·a = a+a
  (-a)+a = 0
  using prems Ring_ZF_1_L1 group0.inverse_in_group group0.group_inv_of_inv
```

```
    group0.group0_2_L6 group0.group0_2_L2 monoid0.group0_1_L3
    Ring_ZF_1_L2 ring_oper_distr
  by auto
```

Properties that require two elements of a ring.

```
lemma (in ring0) Ring_ZF_1_L4: assumes A1: a∈R b∈R
  shows
  a+b ∈ R
  a-b ∈ R
  a·b ∈ R
  a+b = b+a
  using ringAssum prems Ring_ZF_1_L1 Ring_ZF_1_L3
    group0.group0_2_L1 monoid0.group0_1_L1
    IsAring_def IsCommutative_def
  by auto
```

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 -
  def D1: a ≡ x·1
  def D2: b ≡ x·0
  def D3: c ≡ 1·x
  def D4: d ≡ 0·x
  from A1 D1 D2 D3 D4 have
    a + b = x·(1 + 0)   c + d = (1 + 0)·x
    using Ring_ZF_1_L2 ring_oper_distr by auto
  moreover from D1 D3 have x·(1 + 0) = a (1 + 0)·x = c
    using Ring_ZF_1_L2 Ring_ZF_1_L3 by auto
  ultimately have a + b = a and T1: c + d = c by auto
  moreover from A1 D1 D2 D3 D4 have
    a ∈ R  b ∈ R and T2: c ∈ R  d ∈ R
    using Ring_ZF_1_L2 Ring_ZF_1_L4 by auto
  ultimately have b = 0 using
    Ring_ZF_1_L1 group0.group0_2_L7 by simp
  moreover from T2 T1 have d = 0 using
    Ring_ZF_1_L1 group0.group0_2_L7 by simp
  moreover from D2 D4 have b = x·0  d = 0·x by auto
  ultimately show x·0 = 0  0·x = 0 by auto
qed
```

Negative can be pulled out of a product.

```
lemma (in ring0) Ring_ZF_1_L7:
  assumes A1: a∈R  b∈R
  shows
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  (-a)·b = a·(-b)
proof -
```

```
  from A1 have I:
    a·b ∈ R (-a) ∈ R ((-a)·b) ∈ R
    (-b) ∈ R a·(-b) ∈ R
    using Ring_ZF_1_L3 Ring_ZF_1_L4 by auto
  moreover have (-a)·b + a·b = 0
    and II: a·(-b) + a·b = 0
  proof -
    from A1 I have
      (-a)·b + a·b = ((-a)+ a)·b
      a·(-b) + a·b= a·((-b)+b)
      using ring_oper_distr by auto
    moreover from A1 have
      ((-a)+ a)·b = 0
      a·((-b)+b) = 0
      using Ring_ZF_1_L1 group0.group0_2_L6 Ring_ZF_1_L6
      by auto
    ultimately show
      (-a)·b + a·b = 0
      a·(-b) + a·b = 0
      by auto
  qed
  ultimately show (-a)·b = -(a·b)
    using Ring_ZF_1_L1 group0.group0_2_L9 by simp
  moreover from I II show a·(-b) = -(a·b)
    using Ring_ZF_1_L1 group0.group0_2_L9 by simp
  ultimately show (-a)·b = a·(-b) by simp
qed
```

Minus times minus is plus.

```
lemma (in ring0) Ring_ZF_1_L7A: assumes a∈R  b∈R
  shows (-a)·(-b) = a·b
  using prems Ring_ZF_1_L3 Ring_ZF_1_L7 Ring_ZF_1_L4
  by simp
```

Subtraction is distributive with respect to multiplication.

```
lemma (in ring0) Ring_ZF_1_L8: assumes a∈R  b∈R  c∈R
  shows
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using prems Ring_ZF_1_L3 ring_oper_distr Ring_ZF_1_L7 Ring_ZF_1_L4
  by auto
```

Other basic properties involving two elements of a ring.

```
lemma (in ring0) Ring_ZF_1_L9: assumes a∈R  b∈R
  shows
  (-b)-a = (-a)-b
  (-(a+b)) = (-a)-b
  (-(a-b)) = ((-a)+b)
  a-(-b) = a+b
```

```
  using prems ringAssum IsAring_def
    Ring_ZF_1_L1 group0.group0_4_L4  group0.group_inv_of_inv
  by auto
```

If the difference of two element is zero, then those elements are equal.

```
lemma (in ring0) Ring_ZF_1_L9A:
  assumes A1: a∈R  b∈R and A2: a-b = 0
  shows a=b
proof -
  from A1 A2 have
    group0(R,A)
    a∈R  b∈R
    A⟨a,GroupInv(R,A)(b)⟩ = TheNeutralElement(R,A)
    using Ring_ZF_1_L1 by auto
  then show a=b by (rule group0.group0_2_L11A)
qed
```

Other basic properties involving three elements of a ring.

```
lemma (in ring0) Ring_ZF_1_L10:
  assumes a∈R  b∈R  c∈R
  shows
  a+(b+c) = a+b+c

  a-(b+c) = a-b-c
  a-(b-c) = a-b+c
  using prems ringAssum Ring_ZF_1_L1 group0.group_oper_assoc
    IsAring_def group0.group0_4_L4A by auto
```

Another property with three elements.

```
lemma (in ring0) Ring_ZF_1_L10A:
  assumes A1: a∈R  b∈R  c∈R
  shows a+(b-c) = a+b-c
  using prems Ring_ZF_1_L3 Ring_ZF_1_L10 by simp
```

Associativity of addition and multiplication.

```
lemma (in ring0) Ring_ZF_1_L11:
  assumes a∈R  b∈R  c∈R
  shows
  a+b+c = a+(b+c)
  a·b·c = a·(b·c)
  using prems ringAssum Ring_ZF_1_L1 group0.group_oper_assoc
    IsAring_def IsAmonoid_def IsAssociative_def
  by auto
```

An interpretation of what it means that a ring has no zero divisors.

```
lemma (in ring0) Ring_ZF_1_L12:
  assumes HasNoZeroDivs(R,A,M)
  and a∈R  a≠0  b∈R  b≠0
```

```
  shows a·b≠0
  using prems HasNoZeroDivs_def by auto
```

In rings with no zero divisors we can cancel nonzero factors.

```
lemma (in ring0) Ring_ZF_1_L12A:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a∈R  b∈R  c∈R
  and A3: a·c = b·c and A4: c≠0
  shows a=b
proof -
  from A2 have T: a·c ∈ R  a-b ∈ R
    using Ring_ZF_1_L4 by auto
  with A1 A2 A3 have a-b = 0 ∨ c=0
    using Ring_ZF_1_L3 Ring_ZF_1_L8 HasNoZeroDivs_def
    by simp
  with A2 A4 have a∈R  b∈R  a-b = 0
    by auto
  then show a=b by (rule Ring_ZF_1_L9A)
qed
```

In rings with no zero divisors if two elements are different, then after multiplying by a nonzero element they are still different.

```
lemma (in ring0) Ring_ZF_1_L12B:
  assumes A1: HasNoZeroDivs(R,A,M)
  a∈R   b∈R   c∈R   a≠b   c≠0
  shows  a·c ≠ b·c
  using A1 Ring_ZF_1_L12A by auto
```

In rings with no zero divisors multiplying a nonzero element by a nonone element changes the value.

```
lemma (in ring0) Ring_ZF_1_L12C:
  assumes A1: HasNoZeroDivs(R,A,M) and
  A2: a∈R  b∈R and A3: 0≠a  1≠b
  shows a ≠ a·b
proof -
  { assume a = a·b
    with A1 A2 have a = 0 ∨ b-1 = 0
      using Ring_ZF_1_L3 Ring_ZF_1_L2 Ring_ZF_1_L8
        Ring_ZF_1_L3 Ring_ZF_1_L2 Ring_ZF_1_L4 HasNoZeroDivs_def
      by simp
    with A2 A3 have False
      using Ring_ZF_1_L2 Ring_ZF_1_L9A by auto
  } then show a ≠ a·b by auto
qed
```

If a square is nonzero, then the element is nonzero.

```
lemma (in ring0) Ring_ZF_1_L13:
  assumes a∈R  and a² ≠ 0
  shows a≠0
```

```
  using prems Ring_ZF_1_L2 Ring_ZF_1_L6 by auto
```

Square of an element and its opposite are the same.

```
lemma (in ring0) Ring_ZF_1_L14:
  assumes a∈R shows (-a)² = ((a)²)
  using prems Ring_ZF_1_L7A by simp
```

Adding zero to a set that is closed under addition results in a set that is also closed under addition. This is a property of groups.

```
lemma (in ring0) Ring_ZF_1_L15:
  assumes H ⊆ R and H {is closed under} A
  shows (H ∪ {0}) {is closed under} A
  using prems Ring_ZF_1_L1 group0.group0_2_L17 by simp
```

Adding zero to a set that is closed under multiplication results in a set that is also closed under multiplication.

```
lemma (in ring0) Ring_ZF_1_L16:
  assumes A1: H ⊆ R and A2: H {is closed under} M
  shows (H ∪ {0}) {is closed under} M
  using prems Ring_ZF_1_L2 Ring_ZF_1_L6 IsOpClosed_def
  by auto
```

The ring is trivial iff $0 = 1$.

```
lemma (in ring0) Ring_ZF_1_L17: shows R = {0} ⟷ 0=1
proof
  assume R = {0}
  then show 0=1 using Ring_ZF_1_L2
    by blast
next assume A1: 0 = 1
  then have R ⊆ {0}
    using Ring_ZF_1_L3 Ring_ZF_1_L6 by auto
  moreover have {0} ⊆ R using Ring_ZF_1_L2 by auto
  ultimately show R = {0} by auto
qed
```

The sets $\{m \cdot x.x \in R\}$ and $\{-m \cdot x.x \in R\}$ are the same.

```
lemma (in ring0) Ring_ZF_1_L18: assumes A1: m∈R
  shows {m·x. x∈R} = {(-m)·x. x∈R}
proof
  { fix a assume a ∈ {m·x. x∈R}
    then obtain x where x∈R and a = m·x
      by auto
    with A1 have (-x) ∈ R  and a = (-m)·(-x)
      using Ring_ZF_1_L3 Ring_ZF_1_L7A by auto
    then have a ∈ {(-m)·x. x∈R}
      by auto
  } then show {m·x. x∈R} ⊆ {(-m)·x. x∈R}
    by auto
```

```
next
  { fix a assume a ∈ {(-m)·x. x∈R}
    then obtain x where x∈R and a = (-m)·x
      by auto
    with A1 have (-x) ∈ R and a = m·(-x)
      using Ring_ZF_1_L3 Ring_ZF_1_L7 by auto
    then have a ∈ {m·x. x∈R} by auto
  } then show {(-m)·x. x∈R} ⊆ {m·x. x∈R}
    by auto
qed
```

18.2 Rearrangement lemmas

In happens quite often that we want to show a fact like $(a + b)c + d = (ac+d-e)+(bc+e)$in rings. This is trivial in romantic math and probably there is a way to make it trivial in formalized math. However, I don't know any other way than to tediously prove each such rearrangement when it is needed. This section collects facts of this type.

Rearrangements with two elements of a ring.

```
lemma (in ring0) Ring_ZF_2_L1: assumes a∈R b∈R
  shows a+b·a = (b+1)·a
  using prems Ring_ZF_1_L2 ring_oper_distr Ring_ZF_1_L3 Ring_ZF_1_L4
  by simp
```

Raearrangements with two elements and cancelling.

```
lemma (in ring0) Ring_ZF_2_L1A: assumes a∈R b∈R
  shows
  a-b+b = a
  a+b-a = b
  (-a)+b+a = b
  (-a)+(b+a) = b
  a+(b-a) = b
  using prems Ring_ZF_1_L1 group0.group0_2_L16 group0.group0_4_L6A
  by auto
```

In commutative rings $a-(b+1)c = (a-d-c)+(d-bc)$. For unknown reasons we have to use the raw set notation in the proof, otherwise all methods fail.

```
lemma (in ring0) Ring_ZF_2_L2:
  assumes A1: a∈R  b∈R  c∈R  d∈R
  shows a-(b+1)·c = (a-d-c)+(d-b·c)
proof -
  def D1: B == b·c
  from ringAssum have A {is commutative on} R
    using IsAring_def by simp
  moreover from A1 D1 have a∈R B ∈ R c∈R d∈R
    using Ring_ZF_1_L4 by auto
  ultimately have A⟨a, GroupInv(R,A)(A⟨B, c⟩)⟩ =
```

```
    A⟨A⟨A⟨a, GroupInv(R, A)(d)⟩,GroupInv(R, A)(c)⟩,
    A⟨d,GroupInv(R, A)(B)⟩⟩
    using Ring_ZF_1_L1 group0.group0_4_L8 by blast
  with D1 A1 show thesis
    using Ring_ZF_1_L2 ring_oper_distr Ring_ZF_1_L3 by simp
qed
```

Rerrangement about adding linear functions.

```
lemma (in ring0) Ring_ZF_2_L3:
  assumes A1: a∈R  b∈R  c∈R  d∈R  x∈R
  shows (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
proof -
  from A1 have
    group0(R,A)
    A {is commutative on} R
    a·x ∈ R  b∈R  c·x ∈ R  d∈R
    using Ring_ZF_1_L1 Ring_ZF_1_L4 by auto
  then have A⟨A<a·x,b>,A<c·x,d>⟩ = A⟨A<a·x,c·x>,A<b,d>⟩
    by (rule group0.group0_4_L8)
  with A1 show
    (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
    using ring_oper_distr by simp
qed
```

Rearrangement with three elements

```
lemma (in ring0) Ring_ZF_2_L4:
  assumes M {is commutative on} R
  and a∈R  b∈R  c∈R
  shows a·(b·c) = a·c·b
  using prems IsCommutative_def Ring_ZF_1_L11
  by simp
```

Some other rearrangements with three elements.

```
lemma (in ring0) ring_rearr_3_elemA:
  assumes A1: M {is commutative on} R and
  A2: a∈R  b∈R  c∈R
  shows
  a·(a·c) - b·(-b·c) = (a·a + b·b)·c
  a·(-b·c) + b·(a·c) = 0
proof -
  from A2 have T:
    b·c ∈ R  a·a ∈ R  b·b ∈ R
    b·(b·c) ∈ R  a·(b·c) ∈ R
    using  Ring_ZF_1_L4 by auto
  with A2 show
    a·(a·c) - b·(-b·c) = (a·a + b·b)·c
    using Ring_ZF_1_L7 Ring_ZF_1_L3 Ring_ZF_1_L11
      ring_oper_distr by simp
  from A2 T have
```

```
    a·(-b·c) + b·(a·c) = (-a·(b·c)) + b·a·c
      using Ring_ZF_1_L7 Ring_ZF_1_L11 by simp
  also from A1 A2 T have ... = 0
    using IsCommutative_def Ring_ZF_1_L11 Ring_ZF_1_L3
    by simp
  finally show a·(-b·c) + b·(a·c) = 0
    by simp
qed
```

Some rearrangements with four elements. Properties of abelian groups.

```
lemma (in ring0) Ring_ZF_2_L5:
  assumes a∈R  b∈R  c∈R  d∈R
  shows
  a - b - c - d = a - d - b - c
  a + b + c - d = a - d + b + c
  a + b - c - d = a - c + (b - d)
  a + b + c + d = a + c + (b + d)
  using prems Ring_ZF_1_L1 group0.rearr_ab_gr_4_elemB
    group0.rearr_ab_gr_4_elemA by auto
```

Two big rearranegements with six elements, useful for proving properties of complex addition and multiplication.

```
lemma (in ring0) Ring_ZF_2_L6:
  assumes A1: a∈R  b∈R  c∈R  d∈R  e∈R  f∈R
  shows
  a·(c·e - d·f) - b·(c·f + d·e) =
  (a·c - b·d)·e - (a·d + b·c)·f
  a·(c·f + d·e) + b·(c·e - d·f) =
  (a·c - b·d)·f + (a·d + b·c)·e
  a·(c+e) - b·(d+f) = a·c - b·d + (a·e - b·f)
  a·(d+f) + b·(c+e) = a·d + b·c + (a·f + b·e)
proof -
  from A1 have T:
    c·e ∈ R  d·f ∈ R  c·f ∈ R  d·e ∈ R
    a·c ∈ R  b·d ∈ R  a·d ∈ R  b·c ∈ R
    b·f ∈ R  a·e ∈ R  b·e ∈ R  a·f ∈ R
    a·c·e ∈ R  a·d·f ∈ R
    b·c·f ∈ R  b·d·e ∈ R
    b·c·e ∈ R  b·d·f ∈ R
    a·c·f ∈ R  a·d·e ∈ R
    a·c·e - a·d·f ∈ R
    a·c·e - b·d·e ∈ R
    a·c·f + a·d·e ∈ R
    a·c·f - b·d·f ∈ R
    a·c + a·e ∈ R
    a·d + a·f ∈ R
    using Ring_ZF_1_L4 by auto
  with A1 show a·(c·e - d·f) - b·(c·f + d·e) =
    (a·c - b·d)·e - (a·d + b·c)·f
```

```
    using Ring_ZF_1_L8 ring_oper_distr Ring_ZF_1_L11
      Ring_ZF_1_L10 Ring_ZF_2_L5 by simp
  from A1 T show
    a·(c·f + d·e) + b·(c·e - d·f) =
    (a·c - b·d)·f + (a·d + b·c)·e
    using Ring_ZF_1_L8 ring_oper_distr Ring_ZF_1_L11
    Ring_ZF_1_L10A Ring_ZF_2_L5 Ring_ZF_1_L10
    by simp
  from A1 T show
    a·(c+e) - b·(d+f) = a·c - b·d + (a·e - b·f)
    a·(d+f) + b·(c+e) = a·d + b·c + (a·f + b·e)
    using ring_oper_distr Ring_ZF_1_L10 Ring_ZF_2_L5
    by auto
qed

end
```

19 Ring_ZF_1.thy

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.

19.1 The ring of classes of almost homomorphisms

Almost homomorphisms do not form a ring as the regular homomorphisms do because the lifted group operation is not distributive with respect to composition – we have $s \circ (r \cdot q) \neq s \circ r \cdot s \circ q$ in general. However, we do have $s \circ (r \cdot q) \approx s \circ r \cdot s \circ q$ in the sense of the equivalence relation defined by the group of finite range functions (that is a normal subgroup of almost homomorphisms, if the group is abelian). This allows to define a natural ring structure on the classes of almost homomorphisms.

The next lemma provides a formula useful for proving that two sides of the distributive law equation for almost homomorphisms are almost equal.

```
lemma (in group1) Ring_ZF_1_1_L1:
  assumes A1: s∈AH r∈AH q∈AH and A2: n∈G
  shows
  ((s∘(r·q))(n))·(((s∘r)·(s∘q))(n))⁻¹= δ(s,<r(n),q(n)>)
  ((r·q)∘s)(n) = ((r∘s)·(q∘s))(n)
proof -
  from groupAssum isAbelian A1 have T1:
    r·q ∈ AH s∘r ∈ AH  s∘q ∈ AH (s∘r)·(s∘q) ∈ AH
    r∘s ∈ AH q∘s ∈ AH  (r∘s)·(q∘s) ∈ AH
    using Group_ZF_3_2_L15 Group_ZF_3_4_T1 by auto
  from A1 A2 have T2: r(n) ∈ G q(n) ∈ G s(n) ∈ G
    s(r(n)) ∈ G s(q(n)) ∈ G δ(s,<r(n),q(n)>) ∈ G
    s(r(n))·s(q(n)) ∈ G r(s(n)) ∈ G q(s(n)) ∈ G
    r(s(n))·q(s(n)) ∈ G
    using AlmostHoms_def apply_funtype Group_ZF_3_2_L4B
    group0_2_L1 monoid0.group0_1_L1 by auto
  with T1 A1 A2 isAbelian show
    ((s∘(r·q))(n))·(((s∘r)·(s∘q))(n))⁻¹= δ(s,<r(n),q(n)>)
    ((r·q)∘s)(n) = ((r∘s)·(q∘s))(n)
    using Group_ZF_3_2_L12 Group_ZF_3_4_L2 Group_ZF_3_4_L1 group0_4_L6A
    by auto
qed
```

The sides of the distributive law equations for almost homomorphisms are almost equal.

```
lemma (in group1) Ring_ZF_1_1_L2:
```

```
  assumes A1: s∈AH r∈AH q∈AH
  shows
  s∘(r·q) ≈ (s∘r)·(s∘q)
  (r·q)∘s = (r∘s)·(q∘s)
proof -
  from A1 have ∀n∈G. <r(n),q(n)> ∈ G×G
    using AlmostHoms_def apply_funtype by auto
  moreover from A1 have {δ(s,x). x ∈ G×G} ∈ Fin(G)
    using AlmostHoms_def by simp
  ultimately have {δ(s,<r(n),q(n)>). n∈G} ∈ Fin(G)
    by (rule Finite1_L6B)
  with A1 have
    {((s∘(r·q))(n))·(((s∘r)·(s∘q))(n))⁻¹. n ∈ G} ∈ Fin(G)
    using Ring_ZF_1_1_L1 by simp
  moreover from groupAssum isAbelian A1 A1 have
    s∘(r·q) ∈ AH (s∘r)·(s∘q) ∈ AH
    using Group_ZF_3_2_L15 Group_ZF_3_4_T1 by auto
  ultimately show s∘(r·q) ≈ (s∘r)·(s∘q)
    using Group_ZF_3_4_L12 by simp
  from groupAssum isAbelian A1 have
    (r·q)∘s : G→G (r∘s)·(q∘s) : G→G
    using Group_ZF_3_2_L15 Group_ZF_3_4_T1 AlmostHoms_def
    by auto
  moreover from A1 have
    ∀n∈G. ((r·q)∘s)(n) = ((r∘s)·(q∘s))(n)
    using Ring_ZF_1_1_L1 by simp
  ultimately show (r·q)∘s = (r∘s)·(q∘s)
    using fun_extension_iff by simp
qed
```

The essential condition to show the distributivity for the operations defined on classes of almost homomorphisms.

```
lemma (in group1) Ring_ZF_1_1_L3:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  and A2: a ∈ AH//R b ∈ AH//R c ∈ AH//R
  and A3: A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows M⟨a,A<b,c>⟩ = A⟨M<a,b>,M<a,c>⟩ ∧
  M⟨A<b,c>,a⟩ = A⟨M<b,a>,M<c,a>⟩
proof
  from A2 obtain s q r where D1: s∈AH r∈AH q∈AH
    a = R{s} b = R{q} c = R{r}
    using quotient_def by auto
  from A1 have T1:equiv(AH,R)
      using Group_ZF_3_3_L3 by simp
  with A1 A3 D1 groupAssum isAbelian have
    M< a,A<b,c> > = R{s∘(q·r)}
    using Group_ZF_3_3_L4 EquivClass_1_L10
    Group_ZF_3_2_L15 Group_ZF_3_4_L13A by simp
  also have R{s∘(q·r)} = R{(s∘q)·(s∘r)}
```

```
  proof -
    from T1 D1 have equiv(AH,R) s∘(q·r)≈(s∘q)·(s∘r)
      using Ring_ZF_1_1_L2 by auto
    with A1 show thesis using equiv_class_eq by simp
  qed
  also from A1 T1 D1 A3 have
    R{(s∘q)·(s∘r)} = A⟨M<a,b>,M<a,c>⟩
    using Group_ZF_3_3_L4 Group_ZF_3_4_T1 EquivClass_1_L10
    Group_ZF_3_3_L3 Group_ZF_3_4_L13A EquivClass_1_L10 Group_ZF_3_4_T1
    by simp
  finally show M⟨a,A<b,c>⟩ = A⟨M<a,b>,M<a,c>⟩ by simp
  from A1 A3 T1 D1 groupAssum isAbelian show
    M⟨A<b,c>,a⟩ = A⟨M<b,a>,M<c,a>⟩
    using Group_ZF_3_3_L4 EquivClass_1_L10 Group_ZF_3_4_L13A
      Group_ZF_3_2_L15 Ring_ZF_1_1_L2 Group_ZF_3_4_T1 by simp
qed
```

The projection of the first group operation on almost homomorphisms is distributive with respect to the second group operation.

```
lemma (in group1) Ring_ZF_1_1_L4:
  assumes A1: R = QuotientGroupRel(AH,Op1,FR)
  and A2: A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows IsDistributive(AH//R,A,M)
proof -
  from A1 A2 have ∀a∈(AH//R).∀b∈(AH//R).∀c∈(AH//R).
  M⟨a,A<b,c>⟩ = A⟨M<a,b>, M<a,c>⟩ ∧
  M⟨A<b,c>, a⟩ = A⟨M<b,a>,M<c,a>⟩
    using Ring_ZF_1_1_L3 by simp
  then show thesis using IsDistributive_def by simp
qed
```

The classes of almost homomorphisms form a ring.

```
theorem (in group1) Ring_ZF_1_1_T1:
  assumes R = QuotientGroupRel(AH,Op1,FR)
  and A = ProjFun2(AH,R,Op1) M = ProjFun2(AH,R,Op2)
  shows IsAring(AH//R,A,M)
  using prems QuotientGroupOp_def Group_ZF_3_3_T1 Group_ZF_3_4_T2
    Ring_ZF_1_1_L4 IsAring_def by simp

end
```

20 OrderedRing_ZF.thy

theory `OrderedRing_ZF` **imports** `Ring_ZF OrderedGroup_ZF`

begin

In this theory file we consider ordered rings.

20.1 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.

constdefs

```
IsAnOrdRing(R,A,M,r) ≡
( IsAring(R,A,M) ∧ (M {is commutative on} R) ∧
r⊆R×R ∧ IsLinOrder(R,r) ∧
(∀a b. ∀ c∈R. <a,b> ∈ r ⟶ ⟨A<a,c>,A<b,c>⟩ ∈ r) ∧
(Nonnegative(R,A,r) {is closed under} M))
```

The next context (locale) defines notation used for ordered rings. We do that by extending the notation defined in the `ring0` locale and adding some assumptions to make sure we are talking about ordered rings in this context.

locale `ring1 = ring0 +`

assumes `mult_commut: M {is commutative on} R`

fixes `r`

assumes `ordincl: r ⊆ R×R`

assumes `linord: IsLinOrder(R,r)`

fixes `lesseq` **(infix** `≤ 68)`
defines `lesseq_def [simp]: a ≤ b ≡ <a,b> ∈ r`

fixes `sless` **(infix** `< 68)`
defines `sless_def [simp]: a < b ≡ a≤b ∧ a≠b`

assumes `ordgroup: ∀a b. ∀ c∈R. a≤b ⟶ a+c ≤ b+c`

assumes `pos_mult_closed: Nonnegative(R,A,r) {is closed under} M`

fixes `abs (| _ |)`

```
  defines abs_def [simp]: |a| ≡ AbsoluteValue(R,A,r)(a)

  fixes positiveset (R+)
  defines positiveset_def [simp]: R+ ≡ PositiveSet(R,A,r)
```

The next lemma assures us that we are talking about ordered rings in the `ring1` context.

```
lemma (in ring1) OrdRing_ZF_1_L1: shows IsAnOrdRing(R,A,M,r)
  using ring0_def ringAssum mult_commut ordincl linord ordgroup
    pos_mult_closed IsAnOrdRing_def by simp
```

We can use theorems proven in the `ring1` context whenever we talk about an ordered ring.

```
lemma OrdRing_ZF_1_L2: assumes IsAnOrdRing(R,A,M,r)
  shows ring1(R,A,M,r)
  using prems IsAnOrdRing_def ring1_axioms.intro ring0_def ring1_def
  by simp
```

In the `ring1` context $a \leq b$ implies that a, b are elements of the ring.

```
lemma (in  ring1) OrdRing_ZF_1_L3: assumes a≤b
  shows a∈R  b∈R
  using prems ordincl by auto
```

Ordered ring is an ordered group, hence we can use theorems proven in the `group3` context.

```
lemma (in  ring1) OrdRing_ZF_1_L4: shows
  IsAnOrdGroup(R,A,r)
  r {is total on} R
  A {is commutative on} R
  group3(R,A,r)
proof -
  { fix a b g assume A1: g∈R and A2: a≤b
    with ordgroup have a+g ≤ b+g
      by simp
    moreover from ringAssum A1 A2 have
      a+g = g+a  b+g = g+b
      using OrdRing_ZF_1_L3 IsAring_def IsCommutative_def by auto
    ultimately have
      a+g ≤ b+g  g+a ≤ g+b
      by auto
  } hence
    ∀g∈R. ∀a b. a≤b ⟶ a+g ≤ b+g ∧ g+a ≤ g+b
    by simp
  with ringAssum ordincl linord show
    IsAnOrdGroup(R,A,r)
    group3(R,A,r)
    r {is total on} R
    A {is commutative on} R
```

```
    using IsAring_def Order_ZF_1_L2 IsAnOrdGroup_def group3_def IsLinOrder_def
    by auto
qed
```

The order relation in rings is transitive.

```
lemma (in ring1) ring_ord_transitive: assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    group3(R,A,r)  ⟨a,b⟩ ∈ r   ⟨b,c⟩ ∈ r
    using OrdRing_ZF_1_L4 by auto
  then have ⟨a,c⟩ ∈ r by (rule group3.Group_order_transitive)
  then show  a≤c by simp
qed
```

Transitivity for the strict order: if $a < b$ and $b \leq c$, then $a < c$. Property of ordered groups.

```
lemma (in ring1) ring_strict_ord_trans:
  assumes A1: a<b and A2: b≤c
  shows a<c
proof -
  from A1 A2 have
    group3(R,A,r)
    ⟨a,b⟩ ∈ r ∧ a≠b  ⟨b,c⟩ ∈ r
    using OrdRing_ZF_1_L4 by auto
    then have ⟨a,c⟩ ∈ r ∧ a≠c by (rule group3.OrderedGroup_ZF_1_L4A)
    then show a<c by simp
qed
```

Another version of transitivity for the strict order: if $a \leq b$ and $b < c$, then $a < c$. Property of ordered groups.

```
lemma (in ring1) ring_strict_ord_transit:
  assumes A1: a≤b and A2: b<c
  shows a<c
proof -
  from A1 A2 have
    group3(R,A,r)
    ⟨a,b⟩ ∈ r  ⟨b,c⟩ ∈ r ∧ b≠c
    using OrdRing_ZF_1_L4 by auto
  then have ⟨a,c⟩ ∈ r ∧ a≠c by (rule group3.group_strict_ord_transit)
  then show a<c by simp
qed
```

The next lemma shows what happens when one element of an ordered ring is not greater or equal than another.

```
lemma (in ring1) OrdRing_ZF_1_L4A: assumes A1: a∈R  b∈R
  and A2: ¬(a≤b)
  shows b ≤ a  (-a) ≤ (-b)  a≠b
```

```
proof -
  from A1 A2 have I:
    group3(R,A,r)
    r {is total on} R
    a ∈ R  b ∈ R  ⟨a, b⟩ ∉ r
    using OrdRing_ZF_1_L4 by auto
  then have ⟨b,a⟩ ∈ r by (rule  group3.OrderedGroup_ZF_1_L8)
  then show b ≤ a by simp
  from I have ⟨GroupInv(R,A)(a),GroupInv(R,A)(b)⟩ ∈ r
    by (rule  group3.OrderedGroup_ZF_1_L8)
  then show  (-a) ≤ (-b) by simp
  from I show a≠b by (rule group3.OrderedGroup_ZF_1_L8)
qed
```

A special case of `OrdRing_ZF_1_L4A` when one of the constants is 0. This is useful for many proofs by cases.

```
corollary (in ring1) ord_ring_split2: assumes A1: a∈R
  shows a≤0 ∨ (0≤a ∧ a≠0)
proof -
  { from A1 have  I: a∈R  0∈R
      using Ring_ZF_1_L2 by auto
    moreover assume A2: ¬(a≤0)
    ultimately have 0≤a by (rule OrdRing_ZF_1_L4A)
    moreover from I A2 have a≠0 by (rule OrdRing_ZF_1_L4A)
    ultimately have 0≤a ∧ a≠0 by simp}
  then show thesis by auto
qed
```

Taking minus on both sides reverses an inequality.

```
lemma (in ring1) OrdRing_ZF_1_L4B: assumes a≤b
  shows (-b) ≤ (-a)
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5
  by simp
```

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
  using pos_mult_closed prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L2
  IsOpClosed_def by simp
```

Double nonnegative is nonnegative.

```
lemma (in  ring1) OrdRing_ZF_1_L5A: assumes A1: 0≤a
  shows 0≤2·a
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5G
  OrdRing_ZF_1_L3 Ring_ZF_1_L3 by simp
```

A sufficient (somewhat redundant) condition for a structure to be an ordered ring. It says that a commutative ring that is a totally ordered group with respect to the additive operation such that set of nonnegative elements is closed under multiplication, is an ordered ring.

```
lemma OrdRing_ZF_1_L6:
  assumes
  IsAring(R,A,M)
  M {is commutative on} R
  Nonnegative(R,A,r) {is closed under} M
  IsAnOrdGroup(R,A,r)
  r {is total on} R
  shows IsAnOrdRing(R,A,M,r)
  using prems IsAnOrdGroup_def Order_ZF_1_L3 IsAnOrdRing_def
  by simp
```

$a \leq b$ iff $a - b \leq 0$. This is a fact from `OrderedGroup.thy`, where it is stated in multiplicative notation.

```
lemma (in ring1) OrdRing_ZF_1_L7:
  assumes a∈R  b∈R
  shows a≤b ⟷ a-b ≤ 0
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9
  by simp
```

Negative times positive is negative.

```
lemma (in ring1) OrdRing_ZF_1_L8:
  assumes A1: a≤0  and A2: 0≤b
  shows a·b ≤ 0
proof -
  from A1 A2 have T1: a∈R  b∈R  a·b ∈ R
    using OrdRing_ZF_1_L3 Ring_ZF_1_L4 by auto
  from A1 A2 have 0≤(-a)·b
    using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5A OrdRing_ZF_1_L5
    by simp
  with T1 show a·b ≤ 0
    using Ring_ZF_1_L7 OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5AA
    by simp
qed
```

We can multiply both sides of an inequality by a nonnegative ring element. This property is sometimes (not here) used to define ordered rings.

```
lemma (in ring1) OrdRing_ZF_1_L9:
  assumes A1: a≤b and A2: 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  from A1 A2 have T1:
    a∈R  b∈R  c∈R  a·c ∈ R  b·c ∈ R
```

```
    using OrdRing_ZF_1_L3 Ring_ZF_1_L4 by auto
  with A1 A2 have (a-b)·c ≤ 0
    using OrdRing_ZF_1_L7 OrdRing_ZF_1_L8 by simp
  with T1 show a·c ≤ b·c
    using Ring_ZF_1_L8 OrdRing_ZF_1_L7 by simp
  with mult_commut T1 show c·a ≤ c·b
    using IsCommutative_def by simp
qed
```

A special case of OrdRing_ZF_1_L9: we can multiply an inequality by a positive ring element.

```
lemma (in ring1) OrdRing_ZF_1_L9A:
  assumes A1: a≤b and A2: c∈R+
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  from A2 have 0 ≤ c using PositiveSet_def
    by simp
  with A1 show a·c ≤ b·c  c·a ≤ c·b
    using OrdRing_ZF_1_L9 by auto
qed
```

A square is nonnegative.

```
lemma (in ring1) OrdRing_ZF_1_L10:
  assumes A1: a∈R shows 0≤(a²)
proof (cases 0≤a)
  assume 0≤a
  then show 0≤(a²) using OrdRing_ZF_1_L5
    by simp
next assume ¬(0≤a)
  with A1 have 0≤((-a)²)
    using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L8A
      OrdRing_ZF_1_L5 by simp
  with A1 show 0≤(a²) using Ring_ZF_1_L14
    by simp
qed
```

1 is nonnegative.

```
corollary (in ring1) ordring_one_is_nonneg: shows 0 ≤ 1
proof -
  have 0 ≤ (1²) using Ring_ZF_1_L2 OrdRing_ZF_1_L10
    by simp
  then show 0 ≤ 1 using Ring_ZF_1_L2 Ring_ZF_1_L3
    by simp
qed
```

In nontrivial rings one is positive.

```
lemma (in ring1) ordring_one_is_pos: assumes 0≠1
```

```
  shows 1 ∈ R+
  using prems Ring_ZF_1_L2 ordring_one_is_nonneg PositiveSet_def
  by auto
```

Nonnegative is not negative. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_1_L11: assumes 0≤a
  shows ¬(a≤0 ∧ a≠0)
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5AB
  by simp
```

A negative element cannot be a square.

```
lemma (in ring1) OrdRing_ZF_1_L12:
  assumes A1: a≤0  a≠0
  shows ¬(∃b∈R. a = (b²))
proof -
  { assume ∃b∈R. a = (b²)
    with A1 have False using OrdRing_ZF_1_L10 OrdRing_ZF_1_L11
      by auto
  } then show thesis by auto
qed
```

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

```
lemma (in ring1) OrdRing_ZF_1_L13: assumes a≤b
  shows 0 ≤ b-a
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9D
  by simp
```

If $a < b$, then $0 < b - a$.

```
lemma (in ring1) OrdRing_ZF_1_L14: assumes a≤b  a≠b
  shows
  0 ≤ b-a  0 ≠ b-a
  b-a ∈ R+
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9E
  by auto
```

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

```
lemma (in ring1) OrdRing_ZF_1_L15:
  assumes a∈R b∈R and 0 ≤ b-a
  shows a≤b
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L9F
  by simp
```

A nonnegative number is does not decrease when multiplied by a number greater or equal 1.

```
lemma (in ring1) OrdRing_ZF_1_L16:
  assumes A1: 0≤a and A2: 1≤b
  shows a≤a·b
proof -
```

```
  from A1 A2 have T: a∈R  b∈R  a·b ∈ R
    using OrdRing_ZF_1_L3 Ring_ZF_1_L4 by auto
  from A1 A2 have 0 ≤ a·(b-1)
    using OrdRing_ZF_1_L13 OrdRing_ZF_1_L5 by simp
  with T show a≤a·b
    using Ring_ZF_1_L8 Ring_ZF_1_L2 Ring_ZF_1_L3 OrdRing_ZF_1_L15
    by simp
qed
```

We can multiply the right hand side of an inequality between nonnegative ring elements by an element greater or equal 1.

```
lemma (in ring1) OrdRing_ZF_1_L17:
  assumes A1: 0≤a and A2: a≤b and A3: 1≤c
  shows a≤b·c
proof -
  from A1 A2 have 0≤b by (rule ring_ord_transitive)
  with A3 have b≤b·c using OrdRing_ZF_1_L16
    by simp
  with A2 show a≤b·c by (rule ring_ord_transitive)
qed
```

Strict order is preserved by translations.

```
lemma (in ring1) ring_strict_ord_trans_inv:
  assumes a<b and c∈R
  shows
  a+c < b+c
  c+a < c+b
  using prems OrdRing_ZF_1_L4 group3.group_strict_ord_transl_inv
  by auto
```

We can put an element on the other side of a strict inequality, changing its sign.

```
lemma (in ring1) OrdRing_ZF_1_L18:
  assumes a∈R  b∈R and  a-b < c
  shows a < c+b
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L12B
  by simp
```

We can add the sides of two inequalities, the first of them strict, and we get a strict inequality. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_1_L19:
  assumes a<b and c≤d
  shows a+c < b+d
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L12C
  by simp
```

We can add the sides of two inequalities, the second of them strict and we get a strict inequality. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_1_L20:
  assumes a≤b and c<d
  shows a+c < b+d
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L12D
  by simp
```

20.2 Absolute value for ordered rings

Absolute value is defined for ordered groups as a function that is the identity on the nonnegative set and the negative of the element (the inverse in the multiplicative notation) on the rest. In this section we consider properties of absolute value related to multiplication in ordered rings.

Absolute value of a product is the product of absolute values: the case when both elements of the ring are nonnegative.

```
lemma (in ring1) OrdRing_ZF_2_L1:
  assumes 0≤a 0≤b
  shows |a·b| = |a|·|b|
  using prems OrdRing_ZF_1_L5 OrdRing_ZF_1_L4
    group3.OrderedGroup_ZF_1_L2 group3.OrderedGroup_ZF_3_L2
  by simp
```

The absolue value of an element and its negative are the same.

```
lemma (in ring1) OrdRing_ZF_2_L2: assumes a∈R
  shows |-a| = |a|
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_3_L7A by simp
```

The next lemma states that $|a \cdot (-b)| = |(-a) \cdot b| = |(-a) \cdot (-b)| = |a \cdot b|$.

```
lemma (in ring1) OrdRing_ZF_2_L3:
  assumes a∈R  b∈R
  shows
  |(-a)·b| = |a·b|
  |a·(-b)| = |a·b|
  |(-a)·(-b)| = |a·b|
  using prems Ring_ZF_1_L4 Ring_ZF_1_L7 Ring_ZF_1_L7A
   OrdRing_ZF_2_L2 by auto
```

This lemma allows to prove theorems for the case of positive and negative elements of the ring separately.

```
lemma (in ring1) OrdRing_ZF_2_L4: assumes a∈R and ¬(0≤a)
  shows 0 ≤ (-a)  0≠a
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L8A
  by auto
```

Absolute value of a product is the product of absolute values.

```
lemma (in ring1) OrdRing_ZF_2_L5:
  assumes A1: a∈R b∈R
```

```
  shows |a·b| = |a|·|b|
proof (cases 0≤a)
  assume A2: 0≤a show |a·b| = |a|·|b|
  proof (cases 0≤b)
    assume 0≤b
    with A2 show |a·b| = |a|·|b|
      using OrdRing_ZF_2_L1 by simp
  next assume ¬(0≤b)
    with A1 A2 have |a·(-b)| = |a|·|-b|
       using OrdRing_ZF_2_L4 OrdRing_ZF_2_L1 by simp
    with A1 show |a·b| = |a|·|b|
      using OrdRing_ZF_2_L2 OrdRing_ZF_2_L3 by simp
  qed
next assume ¬(0≤a)
  with A1 have A3: 0 ≤ (-a)
    using OrdRing_ZF_2_L4 by simp
  show |a·b| = |a|·|b|
  proof (cases 0≤b)
    assume 0≤b
    with A3 have |(-a)·b| = |-a|·|b|
      using OrdRing_ZF_2_L1 by simp
    with A1 show |a·b| = |a|·|b|
      using OrdRing_ZF_2_L2 OrdRing_ZF_2_L3 by simp
  next assume ¬(0≤b)
    with A1 A3 have |(-a)·(-b)| = |-a|·|-b|
      using OrdRing_ZF_2_L4 OrdRing_ZF_2_L1 by simp
    with A1 show |a·b| = |a|·|b|
      using OrdRing_ZF_2_L2 OrdRing_ZF_2_L3 by simp
  qed
qed
```

Triangle inequality. Property of linearly ordered abelian groups.

```
lemma (in ring1) ord_ring_triangle_ineq:  assumes a∈R b∈R
  shows |a+b| ≤ |a|+|b|
  using prems OrdRing_ZF_1_L4 group3.OrdGroup_triangle_ineq
  by simp
```

If $a \leq c$ and $b \leq c$, then $a + b \leq 2 \cdot c$.

```
lemma (in ring1) OrdRing_ZF_2_L6:
  assumes a≤c  b≤c shows a+b ≤ 2·c
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L5B
    OrdRing_ZF_1_L3 Ring_ZF_1_L3 by simp
```

20.3 Positivity in ordered rings

This section is about properties of the set of positive elements R_+.

The set of positive elements is closed under ring addition. This is a property of ordered groups, we just reference a theorem from OrderedGroup_ZF theory in the proof.

```
lemma (in ring1) OrdRing_ZF_3_L1: shows R+ {is closed under} A
  using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L13
  by simp
```

Every element of a ring can be either in the postitive set, equal to zero or its opposite (the additive inverse) is in the positive set. This is a property of ordered groups, we just reference a theorem from `OrderedGroup_ZF` theory.

```
lemma (in ring1) OrdRing_ZF_3_L2: assumes a∈R
  shows Exactly_1_of_3_holds (a=0, a∈R+, (-a) ∈ R+)
  using prems OrdRing_ZF_1_L4 group3.OrdGroup_decomp
  by simp
```

If a ring element $a \neq 0$, and it is not positive, then $-a$ is positive.

```
lemma (in ring1) OrdRing_ZF_3_L2A: assumes a∈R  a≠0  a ∉ R+
  shows (-a) ∈  R+
  using prems OrdRing_ZF_1_L4 group3.OrdGroup_cases
  by simp
```

`R+` is closed under multiplication iff the ring has no zero divisors.

```
lemma (in ring1) OrdRing_ZF_3_L3:
  shows (R+ {is closed under} M)⟷ HasNoZeroDivs(R,A,M)
proof
  assume A1: HasNoZeroDivs(R,A,M)
  { fix a b assume a∈R+  b∈R+
    then have 0≤a  a≠0  0≤b  b≠0
      using PositiveSet_def by auto
    with A1 have a·b ∈ R+
      using OrdRing_ZF_1_L5 Ring_ZF_1_L2 OrdRing_ZF_1_L3 Ring_ZF_1_L12
        OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L2A
      by simp
  } then show  R+ {is closed under} M using IsOpClosed_def
    by simp
next assume A2: R+ {is closed under} M
  { fix a b assume A3: a∈R  b∈R  and a≠0  b≠0
    with A2 have |a·b| ∈ R+
      using OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_3_L12 IsOpClosed_def
        OrdRing_ZF_2_L5 by simp
    with A3 have a·b ≠ 0
      using PositiveSet_def Ring_ZF_1_L4
        OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_3_L2A
      by auto
  } then show HasNoZeroDivs(R,A,M) using HasNoZeroDivs_def
    by auto
qed
```

Another (in addition to `OrdRing_ZF_1_L6` sufficient condition that defines order in an ordered ring starting from the positive set.

```
theorem (in ring0) ring_ord_by_positive_set:
  assumes
  A1: M {is commutative on} R and
  A2: P⊆R  P {is closed under} A  0 ∉ P and
  A3: ∀a∈R. a≠0 ⟶ (a∈P) Xor ((-a) ∈ P) and
  A4: P {is closed under} M and
  A5: r = OrderFromPosSet(R,A,P)
  shows
  IsAnOrdGroup(R,A,r)
  IsAnOrdRing(R,A,M,r)
  r {is total on} R
  PositiveSet(R,A,r) = P
  Nonnegative(R,A,r) = P ∪ {0}
  HasNoZeroDivs(R,A,M)
proof -
  from A2 A3 A5 show
    I: IsAnOrdGroup(R,A,r)  r {is total on} R and
    II: PositiveSet(R,A,r) = P and
    III: Nonnegative(R,A,r) = P ∪ {0}
    using Ring_ZF_1_L1 group0.Group_ord_by_positive_set
    by auto
  from A2 A4 III have Nonnegative(R,A,r) {is closed under} M
    using Ring_ZF_1_L16 by simp
  with ringAssum A1 I show IsAnOrdRing(R,A,M,r)
    using OrdRing_ZF_1_L6 by simp
  with A4 II show HasNoZeroDivs(R,A,M)
    using OrdRing_ZF_1_L2 ring1.OrdRing_ZF_3_L3
    by auto
qed
```

Nontrivial ordered rings are infinite. More precisely we assume that the neutral element of the additive operation is not equal to the multiplicative neutral element and show that the the set of positive elements of the ring is not a finite subset of the ring and the ring is not a finite subset of itself.

```
theorem (in ring1) ord_ring_infinite: assumes 0≠1
  shows
  R+ ∉ Fin(R)
  R ∉ Fin(R)
  using prems Ring_ZF_1_L17 OrdRing_ZF_1_L4 group3.Linord_group_infinite
  by auto

lemma (in ring1) OrdRing_ZF_3_L4:
  assumes 0≠1 and ∀a∈R. ∃b∈B. a≤b
  shows
  ¬IsBoundedAbove(B,r)
  B ∉ Fin(R)
  using prems Ring_ZF_1_L17 OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_2_L2A
  by auto
```

If m is greater or equal the multiplicative unit, then the set $\{m \cdot n : n \in R\}$ is infinite (unless the ring is trivial).

```
lemma (in ring1) OrdRing_ZF_3_L5: assumes A1: 0≠1 and A2: 1≤m
  shows
  {m·x. x∈R+} ∉ Fin(R)
  {m·x. x∈R} ∉ Fin(R)
  {(-m)·x. x∈R} ∉ Fin(R)
proof -
  from A2 have T: m∈R using OrdRing_ZF_1_L3 by simp
  from A2 have 0≤1  1≤m
    using ordring_one_is_nonneg by auto
  then have I: 0≤m by (rule ring_ord_transitive)
  let B = {m·x. x∈R+}
  { fix a assume A3: a∈R
    then have a≤0 ∨ (0≤a ∧ a≠0)
      using ord_ring_split2 by simp
    moreover
    { assume A4: a≤0
      from A1 have m·1 ∈ B using ordring_one_is_pos
        by auto
      with T have m∈B using Ring_ZF_1_L3 by simp
      moreover from A4 I have a≤m by (rule ring_ord_transitive)
      ultimately have ∃b∈B. a≤b by blast }
    moreover
    { assume A4: 0≤a ∧ a≠0
      with A3 have m·a ∈ B using PositiveSet_def
        by auto
      moreover
      from A2 A4 have 1·a ≤ m·a using OrdRing_ZF_1_L9
        by simp
      with A3 have a ≤ m·a using Ring_ZF_1_L3
        by simp
      ultimately have ∃b∈B. a≤b by auto }
    ultimately have ∃b∈B. a≤b by auto
  } then have ∀a∈R. ∃b∈B. a≤b
    by simp
  with A1 show B ∉ Fin(R) using OrdRing_ZF_3_L4
    by simp
  moreover have B ⊆ {m·x. x∈R}
    using PositiveSet_def by auto
  ultimately show {m·x. x∈R} ∉ Fin(R) using Fin_subset
    by auto
  with T show {(-m)·x. x∈R} ∉ Fin(R) using Ring_ZF_1_L18
    by simp
qed
```

If m is less or equal than the negative of multiplicative unit, then the set $\{m \cdot n : n \in R\}$ is infinite (unless the ring is trivial).

```
lemma (in ring1) OrdRing_ZF_3_L6: assumes A1: 0≠1 and A2: m ≤ -1
```

```
  shows {m·x. x∈R} ∉ Fin(R)
proof -
  from A2 have (-(-1)) ≤ -m
    using OrdRing_ZF_1_L4B by simp
  with A1 have {(-m)·x. x∈R} ∉ Fin(R)
    using Ring_ZF_1_L2 Ring_ZF_1_L3 OrdRing_ZF_3_L5
    by simp
  with A2 show {m·x. x∈R} ∉ Fin(R)
    using OrdRing_ZF_1_L3 Ring_ZF_1_L18 by simp
qed
```

All elements greater or equal than an element of R_+ belong to R_+. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_3_L7: assumes A1: a ∈ R+ and A2: a≤b
  shows b ∈ R+
proof -
  from A1 A2 have
    group3(R,A,r)
    a ∈ PositiveSet(R,A,r)
    ⟨a,b⟩ ∈ r
    using OrdRing_ZF_1_L4 by auto
  then have b ∈ PositiveSet(R,A,r)
    by (rule group3.OrderedGroup_ZF_1_L19)
  then show b ∈ R+ by simp
qed
```

A special case of OrdRing_ZF_3_L7: a ring element greater or equal than 1 is positive.

```
corollary (in ring1) OrdRing_ZF_3_L8: assumes A1: 0≠1 and A2: 1≤a
  shows a ∈ R+
proof -
  from A1 A2 have 1 ∈ R+  1≤a
    using ordring_one_is_pos by auto
  then show a ∈ R+ by (rule OrdRing_ZF_3_L7)
qed
```

Adding a positive element to a strictly increases a. Property of ordered groups.

```
lemma (in ring1) OrdRing_ZF_3_L9: assumes A1: a∈R  b∈R+
  shows a ≤ a+b  a ≠ a+b
  using prems OrdRing_ZF_1_L4 group3.OrderedGroup_ZF_1_L22
  by auto
```

A special case of OrdRing_ZF_3_L9: in nontrivial rings adding one to a increases a.

```
corollary (in ring1) OrdRing_ZF_3_L10: assumes A1: 0≠1 and A2: a∈R
  shows a ≤ a+1  a ≠ a+1
  using prems ordring_one_is_pos OrdRing_ZF_3_L9
```

```
  by auto
```

If a is not greater than b, then it is strictly less than $b + 1$.

```
lemma (in ring1) OrdRing_ZF_3_L11: assumes A1: 0≠1 and A2: a≤b
  shows a< b+1
proof -
  from A1 A2 have I: b < b+1
    using OrdRing_ZF_1_L3 OrdRing_ZF_3_L10 by auto
  with A2 show a< b+1 by (rule ring_strict_ord_transit)
qed
```

For any ring element a the greater of a and 1 is a positive element that is greater or equal than m. If we add 1 to it we get a positive element that is strictly greater than m. This holds in nontrivial rings.

```
lemma (in ring1) OrdRing_ZF_3_L12: assumes A1: 0≠1 and A2: a∈R
  shows
  a ≤ GreaterOf(r,1,a)
  GreaterOf(r,1,a) ∈ R+
  GreaterOf(r,1,a) + 1 ∈ R+
  a ≤ GreaterOf(r,1,a) + 1  a ≠ GreaterOf(r,1,a) + 1
proof -
  from linord have r {is total on} R using IsLinOrder_def
    by simp
  moreover from A2 have 1 ∈ R  a∈R
    using Ring_ZF_1_L2 by auto
  ultimately have
    1 ≤ GreaterOf(r,1,a) and
    I: a ≤ GreaterOf(r,1,a)
    using Order_ZF_3_L2 by auto
  with A1 show
    a ≤ GreaterOf(r,1,a) and
    GreaterOf(r,1,a) ∈ R+
    using OrdRing_ZF_3_L8 by auto
  with A1 show GreaterOf(r,1,a) + 1 ∈ R+
    using ordring_one_is_pos OrdRing_ZF_3_L1 IsOpClosed_def
    by simp
  from A1 I show
    a ≤ GreaterOf(r,1,a) + 1  a ≠ GreaterOf(r,1,a) + 1
    using OrdRing_ZF_3_L11 by auto
qed
```

We can multiply strict inequality by a positive element.

```
lemma (in ring1) OrdRing_ZF_3_L13:
  assumes A1: HasNoZeroDivs(R,A,M) and
  A2: a<b and A3: c∈R+
  shows
  a·c < b·c
  c·a < c·b
```

```
proof -
  from A2 A3 have T: a∈R  b∈R  c∈R  c≠0
    using OrdRing_ZF_1_L3 PositiveSet_def by auto
  from A2 A3 have a·c ≤ b·c using OrdRing_ZF_1_L9A
    by simp
  moreover from A1 A2 T have a·c ≠ b·c
    using Ring_ZF_1_L12A by auto
  ultimately show a·c < b·c by simp
  moreover from mult_commut T have a·c = c·a and b·c = c·b
    using IsCommutative_def by auto
  ultimately show c·a < c·b by simp
qed
```

A sufficient condition for an element to be in the set of positive ring elements.

```
lemma (in ring1) OrdRing_ZF_3_L14: assumes 0≤a and a≠0
  shows a ∈ R+
  using prems OrdRing_ZF_1_L3 PositiveSet_def
  by auto
```

If a ring has no zero divisors, the square of a nonzero element is positive.

```
lemma (in ring1) OrdRing_ZF_3_L15:
  assumes HasNoZeroDivs(R,A,M) and a∈R  a≠0
  shows 0 ≤ a²  a² ≠ 0  a² ∈ R+
  using prems OrdRing_ZF_1_L10 Ring_ZF_1_L12 OrdRing_ZF_3_L14
  by auto
```

In rings with no zero divisors we can (strictly) increase a positive element by multiplying it by an element that is greater than 1.

```
lemma (in ring1) OrdRing_ZF_3_L16:
  assumes HasNoZeroDivs(R,A,M) and a ∈ R+ and 1≤b  1≠b
  shows a≤a·b  a ≠ a·b
  using prems PositiveSet_def OrdRing_ZF_1_L16 OrdRing_ZF_1_L3
    Ring_ZF_1_L12C by auto
```

If the right hand side of an inequality is positive we can multiply it by a number that is greater than one.

```
lemma (in ring1) OrdRing_ZF_3_L17:
   assumes A1: HasNoZeroDivs(R,A,M) and A2: b∈R+ and
  A3: a≤b  and A4: 1<c
  shows a<b·c
proof -
  from A1 A2 A4 have b < b·c
    using OrdRing_ZF_3_L16 by auto
  with A3 show a<b·c by (rule ring_strict_ord_transit)
qed
```

We can multiply a right hand side of an inequality between positive numbers by a number that is greater than one.

```
lemma (in ring1) OrdRing_ZF_3_L18:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a ∈ R+ and
  A3: a≤b and A4: 1<c
  shows a<b·c
proof -
  from A2 A3 have b ∈ R+ using OrdRing_ZF_3_L7
    by blast
  with A1 A3 A4 show a<b·c
    using OrdRing_ZF_3_L17 by simp
qed
```

In ordered rings with no zero divisors if at least one of a, b is not zero, then $a^2 + b^2 > 0$, in particular $a^2 + b^2 \neq 0$.

```
lemma (in ring1) OrdRing_ZF_3_L19:
  assumes A1: HasNoZeroDivs(R,A,M) and A2: a∈R  b∈R and
  A3: a ≠ 0 ∨ b ≠ 0
  shows 0 < a² + b²
proof -
  { assume a ≠ 0
    with A1 A2 have 0 ≤ a²  a² ≠ 0
      using OrdRing_ZF_3_L15 by auto
    then have 0 < a² by auto
    moreover from A2 have 0 ≤ b²
      using OrdRing_ZF_1_L10 by simp
    ultimately have 0 + 0 < a² + b²
      using OrdRing_ZF_1_L19 by simp
    hence 0 < a² + b²
      using Ring_ZF_1_L2 Ring_ZF_1_L3 by simp }
  moreover
  { assume A4: a = 0
    then have a² + b² = 0 + b²
      using  Ring_ZF_1_L2 Ring_ZF_1_L6 by simp
    also from A2 have ... = b²
      using Ring_ZF_1_L4 Ring_ZF_1_L3 by simp
    finally have a² + b² = b² by simp
    moreover
    from A3 A4 have b ≠ 0 by simp
    with A1 A2 have 0 ≤ b² and b² ≠ 0
      using OrdRing_ZF_3_L15 by auto
    hence 0 < b² by auto
    ultimately have 0 < a² + b² by simp }
  ultimately show 0 < a² + b²
    by auto
qed
```

```
end
```

21 Field_ZF.thy

theory Field_ZF **imports** Ring_ZF

begin

This theory covers basic facts about fields.

21.1 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.

```
constdefs
  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 +
  assumes mult_commute: M {is commutative on} K

  assumes not_triv: 0 ≠ 1

  assumes inv_exists: ∀a∈K. a≠0 ⟶ (∃b∈K. a·b = 1)

  fixes non_zero (K₀)
  defines non_zero_def[simp]: K₀ ≡ K-{0}

  fixes inv (_⁻¹  [96] 97)
  defines inv_def[simp]: a⁻¹ ≡ GroupInv(K₀,restrict(M,K₀×K₀))(a)
```

The next lemma assures us that we are talking fields in the field0 context.

```
lemma (in field0) Field_ZF_1_L1: shows IsAfield(K,A,M)
  using ringAssum mult_commute not_triv inv_exists IsAfield_def
  by simp
```

We can use theorems proven in the field0 context whenever we talk about a field.

```
lemma Field_ZF_1_L2: assumes IsAfield(K,A,M)
```

```
  shows field0(K,A,M)
  using prems IsAfield_def field0_axioms.intro ring0_def field0_def
  by simp
```

Let's have an explicit statement that the multiplication in fields is commutative.

```
lemma (in field0) field_mult_comm: assumes a∈K  b∈K
  shows a·b = b·a
  using mult_commute prems IsCommutative_def by simp
```

Fields do not have zero divisors.

```
lemma (in field0) field_has_no_zero_divs: shows HasNoZeroDivs(K,A,M)
proof -
  { fix a b assume A1: a∈K  b∈K and A2: a·b = 0  and A3: b≠0
    from inv_exists A1 A3 obtain c where I: c∈K and II: b·c = 1
      by auto
    from A2 have a·b·c = 0·c by simp
    with A1 I have a·(b·c) = 0
      using Ring_ZF_1_L11 Ring_ZF_1_L6 by simp
    with A1 II have a=0 using Ring_ZF_1_L3 by simp }
  then have ∀a∈K.∀b∈K. a·b = 0 ⟶ a=0 ∨ b=0 by auto
    then show thesis using HasNoZeroDivs_def by auto
qed
```

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

```
lemma (in field0) Field_ZF_1_L2: K₀ {is closed under} M
  using Ring_ZF_1_L4 field_has_no_zero_divs Ring_ZF_1_L12
    IsOpClosed_def by auto
```

Any nonzero element has a right inverse that is nonzero.

```
lemma (in field0) Field_ZF_1_L3: assumes A1: a∈K₀
  shows ∃b∈K₀. a·b = 1
proof -
  from inv_exists A1 obtain b where b∈K and a·b = 1
    by auto
  with not_triv A1 show ∃b∈K₀. a·b = 1
    using Ring_ZF_1_L6 by auto
qed
```

If we remove zero, the field with multiplication becomes a group and we can use all theorems proven in group0 context.

```
theorem (in field0) Field_ZF_1_L4: shows
  IsAgroup(K₀,restrict(M,K₀×K₀))
  group0(K₀,restrict(M,K₀×K₀))
  1 = TheNeutralElement(K₀,restrict(M,K₀×K₀))
proof-
  let f = restrict(M,K₀×K₀)
  have
```

```
    M {is associative on} K
    K0 ⊆ K  K0 {is closed under} M
    using Field_ZF_1_L1 IsAfield_def IsAring_def IsAgroup_def
      IsAmonoid_def Field_ZF_1_L2 by auto
  then have f {is associative on} K0
    using func_ZF_4_L3 by simp
  moreover
  from not_triv have
    I: 1∈K0 ∧ (∀a∈K0. f⟨1,a⟩ = a ∧  f⟨a,1⟩ = a)
    using Ring_ZF_1_L2 Ring_ZF_1_L3 by auto
  then have ∃n∈K0. ∀a∈K0. f⟨n,a⟩ = a ∧  f⟨a,n⟩ = a
    by blast
  ultimately have II: IsAmonoid(K0,f) using IsAmonoid_def
    by simp
  then have monoid0(K0,f) using monoid0_def by simp
  moreover note I
  ultimately show 1 = TheNeutralElement(K0,f)
    by (rule monoid0.group0_1_L4)
  then have ∀a∈K0.∃b∈K0. f⟨a,b⟩ =  TheNeutralElement(K0,f)
    using Field_ZF_1_L3 by auto
  with II show IsAgroup(K0,f) by (rule definition_of_group)
  then show group0(K0,f) using group0_def by simp
qed
```

The inverse of a nonzero field element is nonzero.

```
lemma (in field0) Field_ZF_1_L5: assumes A1: a∈K  a≠0
  shows a⁻¹ ∈ K0  (a⁻¹)² ∈ K0   a⁻¹ ∈ K  a⁻¹ ≠ 0
proof -
  from A1 have a ∈ K0 by simp
  then show a⁻¹ ∈ K0 using Field_ZF_1_L4 group0.inverse_in_group
    by auto
  then show  (a⁻¹)² ∈ K0  a⁻¹ ∈ K  a⁻¹ ≠ 0
    using Field_ZF_1_L2 IsOpClosed_def by auto
qed
```

The inverse is really the inverse.

```
lemma (in field0) Field_ZF_1_L6: assumes A1: a∈K  a≠0
  shows a·a⁻¹ = 1  a⁻¹·a = 1
proof -
  let f = restrict(M,K0×K0)
  from A1 have
    group0(K0,f)
    a ∈ K0
    using Field_ZF_1_L4 by auto
  then have
    f⟨a,GroupInv(K0, f)(a)⟩ = TheNeutralElement(K0,f) ∧
    f⟨GroupInv(K0,f)(a),a⟩ = TheNeutralElement(K0, f)
    by (rule group0.group0_2_L6)
  with A1 show a·a⁻¹ = 1  a⁻¹·a = 1
```

```
    using Field_ZF_1_L5 Field_ZF_1_L4 by auto
qed
```

A lemma with two field elements and cancelling.

```
lemma (in field0) Field_ZF_1_L7: assumes a∈K b∈K b≠0
  shows
  a·b·b⁻¹ = a
  a·b⁻¹·b = a
  using prems Field_ZF_1_L5 Ring_ZF_1_L11 Field_ZF_1_L6 Ring_ZF_1_L3
  by auto
```

21.2 Equations and identities

This section deals with more specialized identities that are true in fields.

$a/(a^2) = a$.

```
lemma (in field0) Field_ZF_2_L1: assumes A1: a∈K  a≠0
  shows a·(a⁻¹)² = a⁻¹
proof -
  have a·(a⁻¹)² = a·(a⁻¹·a⁻¹) by simp
  also from A1 have ... =  (a·a⁻¹)·a⁻¹
    using Field_ZF_1_L5 Ring_ZF_1_L11
    by simp
  also from A1 have ... = a⁻¹
    using Field_ZF_1_L6 Field_ZF_1_L5 Ring_ZF_1_L3
    by simp
  finally show a·(a⁻¹)² = a⁻¹ by simp
qed
```

If we multiply two different numbers by a nonzero number, the results will be different.

```
lemma (in field0) Field_ZF_2_L2:
  assumes a∈K  b∈K  c∈K  a≠b  c≠0
  shows a·c⁻¹ ≠ b·c⁻¹
  using prems field_has_no_zero_divs Field_ZF_1_L5 Ring_ZF_1_L12B
  by simp
```

We can put a nonzero factor on the other side of non-identity (is this the best way to call it?) changing it to the inverse.

```
lemma (in field0) Field_ZF_2_L3:
  assumes A1: a∈K  b∈K  b≠0  c∈K   and A2: a·b ≠ c
  shows a ≠ c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ ≠ c·b⁻¹
    using  Ring_ZF_1_L4 Field_ZF_2_L2 by simp
  with A1 show a ≠ c·b⁻¹ using Field_ZF_1_L7
    by simp
qed
```

If if the inverse of b is different than a, then the inverse of a is different than b.

```
lemma (in field0) Field_ZF_2_L4:
  assumes a∈K  a≠0 and b⁻¹ ≠ a
  shows a⁻¹ ≠ b
  using prems Field_ZF_1_L4 group0.group0_2_L11B
  by simp
```

An identity with two field elements, one and an inverse.

```
lemma (in field0) Field_ZF_2_L5:
  assumes a∈K  b∈K b≠0
  shows (1 + a·b)·b⁻¹ = a + b⁻¹
  using prems Ring_ZF_1_L4 Field_ZF_1_L5 Ring_ZF_1_L2 ring_oper_distr

    Field_ZF_1_L7 Ring_ZF_1_L3 by simp
```

An identity with three field elements, inverse and cancelling.

```
lemma (in field0) Field_ZF_2_L6: assumes A1: a∈K  b∈K  b≠0  c∈K
  shows a·b·(c·b⁻¹) = a·c
proof -
  from A1 have T: a·b ∈ K  b⁻¹ ∈ K
    using Ring_ZF_1_L4 Field_ZF_1_L5 by auto
  with mult_commute A1 have a·b·(c·b⁻¹) = a·b·(b⁻¹·c)
    using IsCommutative_def by simp
  moreover
  from A1 T have a·b ∈ K  b⁻¹ ∈ K  c∈K
    by auto
  then have a·b·b⁻¹·c = a·b·(b⁻¹·c)
    by (rule Ring_ZF_1_L11)
  ultimately have a·b·(c·b⁻¹) = a·b·b⁻¹·c by simp
  with A1 show a·b·(c·b⁻¹) = a·c
    using Field_ZF_1_L7 by simp
qed

end
```

22 OrderedField_ZF.thy

theory OrderedField_ZF **imports** OrderedRing_ZF Field_ZF

begin

This theory covers basic facts about ordered fiels.

22.1 Definition and 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.

```
constdefs
  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 oredered rings and adding some assumptions to make sure we are talking about ordered fields in this context. We should rename the carrier from R used in the ring1 context to K, more appriopriate for fields. Theoretically the Isar locale facility supports such renaming, but we experienced diffculties using some lemmas from ring1 locale after renaming.

```
locale field1 = ring1 +

  assumes mult_commute: M {is commutative on} R

  assumes not_triv: 0 ≠ 1

  assumes inv_exists: ∀a∈R. a≠0 ⟶ (∃b∈R. a·b = 1)

  fixes non_zero (R₀)
  defines non_zero_def[simp]: R₀ ≡ R-{0}

  fixes inv (_⁻¹  [96] 97)
  defines inv_def[simp]: a⁻¹ ≡ GroupInv(R₀,restrict(M,R₀×R₀))(a)
```

The next lemma assures us that we are talking fields in the field1 context.

```
lemma (in field1) OrdField_ZF_1_L1: shows IsAnOrdField(R,A,M,r)
  using OrdRing_ZF_1_L1 mult_commute not_triv inv_exists IsAnOrdField_def
```

```
  by simp
```

Ordered field is a field, of course.

```
lemma OrdField_ZF_1_L1A: assumes IsAnOrdField(K,A,M,r)
  shows IsAfield(K,A,M)
  using prems IsAnOrdField_def IsAnOrdRing_def IsAfield_def
  by simp
```

Theorems proven in `field0` (about fields) context are valid in the `field1` context (about ordered fields).

```
lemma (in field1) OrdField_ZF_1_L1B: shows field0(R,A,M)
  using  OrdField_ZF_1_L1 OrdField_ZF_1_L1A Field_ZF_1_L2
  by simp
```

We can use theorems proven in the `field1` context whenever we talk about an ordered field.

```
lemma OrdField_ZF_1_L2: assumes IsAnOrdField(K,A,M,r)
  shows field1(K,A,M,r)
  using prems IsAnOrdField_def OrdRing_ZF_1_L2 ring1_def
    IsAnOrdField_def field1_axioms_def field1_def
  by auto
```

In ordered rings the existence of a right inverse for all positive elements implies the existence of an inverse for all non zero elements.

```
lemma (in ring1) OrdField_ZF_1_L3:
  assumes A1: ∀a∈R₊. ∃b∈R. a·b = 1 and A2: c∈R  c≠0
  shows ∃b∈R. c·b = 1
proof (cases c∈R₊)
  assume c∈R₊
  with A1 show ∃b∈R. c·b = 1 by simp
next assume c∉R₊
  with A2 have (-c) ∈ R₊
    using OrdRing_ZF_3_L2A by simp
  with A1 obtain b where b∈R and (-c)·b = 1
    by auto
  with A2 have (-b) ∈ R  c·(-b) = 1
    using Ring_ZF_1_L3 Ring_ZF_1_L7 by auto
  then show ∃b∈R. c·b = 1 by auto
qed
```

Ordered fields are easier to deal with, because it is sufficient to show the existence of an inverse for the set of positive elements.

```
lemma (in ring1) OrdField_ZF_1_L4:
  assumes 0 ≠ 1 and M {is commutative on} R
  and ∀a∈R₊. ∃b∈R. a·b = 1
  shows IsAnOrdField(R,A,M,r)
  using prems OrdRing_ZF_1_L1 OrdField_ZF_1_L3 IsAnOrdField_def
  by simp
```

The set of positive field elements is closed under multiplication.

```
lemma (in field1) OrdField_ZF_1_L5: shows R+ {is closed under} M
  using OrdField_ZF_1_L1B field0.field_has_no_zero_divs OrdRing_ZF_3_L3
  by simp
```

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

```
lemma (in field1) pos_mul_closed:
  assumes A1: 0 < a  0 < b
  shows 0 < a·b
proof -
  from A1 have a ∈ R+ and  b ∈ R+
    using OrdRing_ZF_3_L14 by auto
  then show 0 < a·b
    using OrdField_ZF_1_L5 IsOpClosed_def PositiveSet_def
    by simp
qed
```

In fields square of a nonzero element is positive.

```
lemma (in field1) OrdField_ZF_1_L6: assumes a∈R  a≠0
  shows a² ∈ R+
  using prems OrdField_ZF_1_L1B field0.field_has_no_zero_divs
    OrdRing_ZF_3_L15 by simp
```

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
  using prems OrdField_ZF_1_L1B field0.Field_ZF_1_L6
  by auto
```

A simple lemma about multiplication and cancelling of a positive field element.

```
lemma (in field1) OrdField_ZF_1_L7A:
  assumes A1: a∈R  b ∈ R+
  shows
  a·b·b⁻¹ = a
  a·b⁻¹·b = a
proof -
  from A1 have b∈R  b≠0 using PositiveSet_def
    by auto
  with A1 show  a·b·b⁻¹ = a and a·b⁻¹·b = a
    using OrdField_ZF_1_L1B field0.Field_ZF_1_L7
    by auto
qed
```

Some properties of the inverse of a positive element.

```
lemma (in field1) OrdField_ZF_1_L8: assumes A1: a ∈ R+
  shows a⁻¹ ∈ R+  a·(a⁻¹) = 1  (a⁻¹)·a = 1
proof -
  from A1 have I: a∈R  a≠0 using PositiveSet_def
    by auto
  with A1 have a·(a⁻¹)² ∈ R+
    using OrdField_ZF_1_L1B field0.Field_ZF_1_L5 OrdField_ZF_1_L6
      OrdField_ZF_1_L5 IsOpClosed_def by simp
  with I show a⁻¹ ∈ R+
    using OrdField_ZF_1_L1B field0.Field_ZF_2_L1
    by simp
  from I show  a·(a⁻¹) = 1  (a⁻¹)·a = 1
    using OrdField_ZF_1_L7 by auto
qed
```

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

```
lemma (in field1) OrdField_ZF_1_L9: assumes a<b
  shows  (b-a)⁻¹ ∈ R+
  using prems OrdRing_ZF_1_L14 OrdField_ZF_1_L8
  by simp
```

In ordered fields if at least one of a, b is not zero, then $a^2 + b^2 > 0$, in particular $a^2 + b^2 \neq 0$ and 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 -
  from A1 A2 show 0 < a² + b²
    using OrdField_ZF_1_L1B field0.field_has_no_zero_divs
      OrdRing_ZF_3_L19 by simp
  then have
    (a² + b²)⁻¹ ∈ R and (a² + b²)·(a² + b²)⁻¹ = 1
    using OrdRing_ZF_1_L3 PositiveSet_def OrdField_ZF_1_L8
    by auto
  then show ∃c∈R. (a² + b²)·c = 1 by auto
qed
```

22.2 Inequalities

In this section we develop tools to deal inequalities in fields.

We can multiply strict inequality by a positive element.

```
lemma (in field1) OrdField_ZF_2_L1:
  assumes a<b and c∈R+
  shows a·c < b·c
  using prems OrdField_ZF_1_L1B field0.field_has_no_zero_divs
    OrdRing_ZF_3_L13
  by simp
```

A special case of OrdField_ZF_2_L1 when we multiply an inverse by an element.

```
lemma (in field1) OrdField_ZF_2_L2:
  assumes A1: a∈R+ and A2: a⁻¹ < b
  shows 1 < b·a
proof -
  from A1 A2 have (a⁻¹)·a < b·a
    using OrdField_ZF_2_L1 by simp
  with A1 show 1 < b·a
    using OrdField_ZF_1_L8 by simp
qed
```

We can multiply an inequality by the inverse of a positive element.

```
lemma (in field1) OrdField_ZF_2_L3:
  assumes a≤b  and c∈R+ shows a·(c⁻¹) ≤ b·(c⁻¹)
  using prems OrdField_ZF_1_L8 OrdRing_ZF_1_L9A
  by simp
```

We can multiply a strict inequality by a positive element or its inverse.

```
lemma (in field1) OrdField_ZF_2_L4:
  assumes a<b and c∈R+
  shows
  a·c < b·c
  c·a < c·b
  a·c⁻¹ < b·c⁻¹
   using prems OrdField_ZF_1_L1B field0.field_has_no_zero_divs
    OrdField_ZF_1_L8 OrdRing_ZF_3_L13 by auto
```

We can put a positive factor on the other side of an inequality, changing it to its inverse.

```
lemma (in field1) OrdField_ZF_2_L5:
  assumes A1: a∈R  b∈R+ and A2: a·b ≤ c
  shows a ≤ c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ ≤ c·b⁻¹
    using OrdField_ZF_2_L3 by simp
  with A1 show a ≤ c·b⁻¹ using OrdField_ZF_1_L7A
    by simp
qed
```

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with a product initially on the right hand side.

```
lemma (in field1) OrdField_ZF_2_L5A:
  assumes A1: b∈R  c∈R+ and A2: a ≤ b·c
  shows a·c⁻¹ ≤ b
proof -
  from A1 A2 have a·c⁻¹ ≤ b·c·c⁻¹
    using OrdField_ZF_2_L3 by simp
```

```
  with A1 show a·c⁻¹ ≤ b using OrdField_ZF_1_L7A
    by simp
qed
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with a product initially on the left hand side.

```
lemma (in field1) OrdField_ZF_2_L6:
  assumes A1: a∈R  b∈R₊ and A2: a·b < c
  shows a < c·b⁻¹
proof -
  from A1 A2 have a·b·b⁻¹ < c·b⁻¹
    using OrdField_ZF_2_L4 by simp
  with A1 show a < c·b⁻¹ using OrdField_ZF_1_L7A
    by simp
qed
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with a product initially on the right hand side.

```
lemma (in field1) OrdField_ZF_2_L6A:
  assumes A1: b∈R  c∈R₊ and A2: a < b·c
  shows a·c⁻¹ < b
proof -
  from A1 A2 have a·c⁻¹ < b·c·c⁻¹
    using OrdField_ZF_2_L4 by simp
  with A1 show a·c⁻¹ < b using OrdField_ZF_1_L7A
    by simp
qed
```

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

```
lemma (in field1) OrdField_ZF_2_L7:
  assumes A1: a∈R₊ and A2: a⁻¹ ≤ b
  shows b⁻¹ ≤ a
proof -
  from A1 have a⁻¹ ∈ R₊ using OrdField_ZF_1_L8
    by simp
  with A2 have b ∈ R₊ using  OrdRing_ZF_3_L7
    by blast
  then have T: b ∈ R₊  b⁻¹ ∈ R₊ using OrdField_ZF_1_L8
    by auto
  with A1 A2 have b⁻¹·a⁻¹·a ≤ b⁻¹·b·a
    using OrdRing_ZF_1_L9A by simp
  moreover
  from A1 A2 T have
    b⁻¹ ∈ R  a∈R a≠0  b∈R  b≠0
    using PositiveSet_def OrdRing_ZF_1_L3 by auto
  then have b⁻¹·a⁻¹·a = b⁻¹ and  b⁻¹·b·a = a
    using OrdField_ZF_1_L1B field0.Field_ZF_1_L7
      field0.Field_ZF_1_L6 Ring_ZF_1_L3
```

```
    by auto
  ultimately show b⁻¹ ≤ a by simp
qed
```

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

```
lemma (in field1) OrdField_ZF_2_L8:
  assumes A1: a∈R₊ and A2: a⁻¹ < b
  shows b⁻¹ < a
proof -
  from A1 A2 have a⁻¹ ∈ R₊  a⁻¹ ≤b
    using OrdField_ZF_1_L8 by auto
  then have b ∈ R₊ using OrdRing_ZF_3_L7
    by blast
  then have b∈R  b≠0 using PositiveSet_def by auto
  with A2 have b⁻¹ ≠ a
    using OrdField_ZF_1_L1B field0.Field_ZF_2_L4
    by simp
  with A1 A2 show b⁻¹ < a
    using OrdField_ZF_2_L7 by simp
qed
```

A technical lemma about solving a strict inequality with three field elements and inverse of a difference.

```
lemma (in field1) OrdField_ZF_2_L9:
  assumes A1: a<b and A2: (b-a)⁻¹ < c
  shows 1 + a·c < b·c
proof -
  from A1 A2 have (b-a)⁻¹ ∈ R₊  (b-a)⁻¹ ≤ c
    using OrdField_ZF_1_L9 by auto
  then have T1: c ∈ R₊ using OrdRing_ZF_3_L7 by blast
  with A1 A2 have T2:
    a∈R  b∈R  c∈R  c≠0   c⁻¹ ∈ R
    using OrdRing_ZF_1_L3 OrdField_ZF_1_L8 PositiveSet_def
    by auto
  with A1 A2  have c⁻¹ + a < b-a + a
    using OrdRing_ZF_1_L14 OrdField_ZF_2_L8 ring_strict_ord_trans_inv
    by simp
  with T1 T2 have (c⁻¹ + a)·c < b·c
    using Ring_ZF_2_L1A OrdField_ZF_2_L1 by simp
  with T1 T2 show 1 + a·c < b·c
    using ring_oper_distr OrdField_ZF_1_L8
    by simp
qed
```

22.3 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.

constdefs

```
  IsAmodelOfReals(K,A,M,r) ≡ IsAnOrdField(K,A,M,r) ∧ (r {is complete})
```

end

23 Int_ZF.thy

theory `Int_ZF` **imports** `OrderedGroup_ZF Finite_ZF_1 Int Nat_ZF`

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.

23.1 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 $\$ \leq$ (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 positive integers using the notion of positive set from the `OrderedGroup` theory.

constdefs

```
  IntegerAddition ≡ { <x,c> ∈ (int×int)×int. fst(x) $+ snd(x) = c}

  IntegerMultiplication ≡
    { <x,c> ∈ (int×int)×int. fst(x) $× snd(x) = c}

  IntegerOrder ≡ {p ∈ int×int. fst(p) $≤ snd(p)}

  PositiveIntegers ≡ PositiveSet(int,IntegerAddition,IntegerOrder)
```

IntegerAddition and IntegerMultiplication are functions on int×int.

```
lemma Int_ZF_1_L1:
  IntegerAddition : int×int → int
  IntegerMultiplication : int×int → int
proof -
  have
    {<x,c> ∈ (int×int)×int. fst(x) $+ snd(x) = c} ∈ int×int→int
    {<x,c> ∈ (int×int)×int. fst(x) $× snd(x) = c} ∈ int×int→int
    using func1_1_L11A by auto
  then show IntegerAddition : int×int → int
    IntegerMultiplication : int×int → int
    using IntegerAddition_def IntegerMultiplication_def by auto
qed
```

The next context (locale) defines notation used for integers. We define **0** to denote the neutral element of addition, **1** as the unit of the multiplicative monoid. We introduce notation `m≤n` for integers and write `m..n` to denote the integer interval with endpoints in m and n. `abs(m)` means the absolute value of m. This is a function defined in `OrderedGroup` that assigns x to itself if x is positive and assigns the opposite of x if $x \leq 0$. Unforunately we cannot use the $|\cdot|$ notation as in the `OrderedGroup` theory as this notation has been hogged by the standard Isabelle's `Int` theory. The notation `-A` where A is a subset of integers means the set $\{-m : m \in A\}$. The symbol `maxf(f,M)` denotes tha maximum of function f over the set A. We also introduce a similar notation for the minimum.

```
locale int0 =

  fixes ints (ℤ)
  defines ints_def [simp]: ℤ ≡ int

  fixes ia (infixl + 69)
  defines ia_def [simp]: a+b ≡ IntegerAddition<a,b>

  fixes iminus :: i⇒i (- _ 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 :: i⇒i (- _ 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 itwo_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 `$+` and `$×` defined in the standard `Int` theory.

```
lemma (in int0) Int_ZF_1_L2: assumes A1: a ∈ ℤ  b ∈ ℤ
  shows
  a+b = a $+ b
  a·b = a $× b
proof -
  let x = <a,b>
  let c = a $+ b
  let d = a $× b
  from A1 have
    <x,c> ∈ {<x,c> ∈ (ℤ×ℤ)×ℤ. fst(x) $+ snd(x) = c}
    <x,d> ∈ {<x,d> ∈ (ℤ×ℤ)×ℤ. fst(x) $× snd(x) = d}
    by auto
  then show a+b = a $+ b  a·b = a $× b
    using IntegerAddition_def IntegerMultiplication_def
      Int_ZF_1_L1 apply_iff by auto
qed
```

Integer addition and multiplication are associative.

```
lemma (in int0) Int_ZF_1_L3:
  assumes x∈ℤ  y∈ℤ  z∈ℤ
  shows x+y+z = x+(y+z)  x·y·z = x·(y·z)
  using prems Int_ZF_1_L2 zadd_assoc zmult_assoc by auto
```

Integer addition and multiplication are commutative.

```
lemma (in int0) Int_ZF_1_L4:
  assumes x∈ℤ  y∈ℤ
  shows x+y = y+x  x·y = y·x
```

```
  using prems Int_ZF_1_L2 zadd_commute zmult_commute
  by auto
```

Zero is neutral for addition and one for multiplication.

```
lemma (in int0) Int_ZF_1_L5: assumes A1:x∈ℤ
  shows ($# 0) + x = x ∧ x + ($# 0) = x
  ($# 1)·x = x ∧ x·($# 1) = x
proof -
  from A1 show ($# 0) + x = x ∧ x + ($# 0) = x
    using  Int_ZF_1_L2 zadd_int0 Int_ZF_1_L4 by simp
  from A1 have ($# 1)·x = x
    using Int_ZF_1_L2 zmult_int1 by simp
  with A1 show ($# 1)·x = x ∧ x·($# 1) = x
    using Int_ZF_1_L4 by simp
qed
```

Zero is neutral for addition and one for multiplication.

```
lemma (in int0) Int_ZF_1_L6: shows ($# 0)∈ℤ ∧
  (∀x∈ℤ. ($# 0)+x = x ∧ x+($# 0) = x)
  ($# 1)∈ℤ ∧
  (∀x∈ℤ. ($# 1)·x = x ∧ x·($# 1) = x)
  using Int_ZF_1_L5 by auto
```

Integers with addition and integers with multiplication form monoids.

```
theorem (in int0) Int_ZF_1_T1: shows
  IsAmonoid(ℤ,IntegerAddition)
  IsAmonoid(ℤ,IntegerMultiplication)
proof -
   have
    ∃e∈ℤ. ∀x∈ℤ. e+x = x ∧ x+e = x
     ∃e∈ℤ. ∀x∈ℤ. e·x = x ∧ x·e = x
     using int0.Int_ZF_1_L6 by auto
   then show IsAmonoid(ℤ,IntegerAddition)
     IsAmonoid(ℤ,IntegerMultiplication) using
     IsAmonoid_def IsAssociative_def Int_ZF_1_L1 Int_ZF_1_L3
     by auto
qed
```

Zero is the neutral element of the integers with addition and one is the neutral element of the integers with multiplication.

```
lemma (in int0) Int_ZF_1_L8: ($# 0) = 0  ($# 1) = 1
proof -
  have monoid0(ℤ,IntegerAddition)
    using Int_ZF_1_T1 monoid0_def by simp
  moreover have
    ($# 0)∈ℤ ∧
    (∀x∈ℤ. IntegerAddition⟨$# 0,x⟩ = x ∧
    IntegerAddition⟨x ,$# 0⟩ = x)
```

```
    using Int_ZF_1_L6 by auto
  ultimately have ($# 0) = TheNeutralElement(ℤ,IntegerAddition)
    by (rule monoid0.group0_1_L4)
  then show ($# 0) = 0 by simp
  have monoid0(int,IntegerMultiplication)
    using Int_ZF_1_T1 monoid0_def by simp
  moreover have ($# 1) ∈ int ∧
    (∀x∈int. IntegerMultiplication⟨$# 1, x⟩ = x ∧
    IntegerMultiplication⟨x ,$# 1⟩ = x)
    using Int_ZF_1_L6 by auto
  ultimately have
    ($# 1) = TheNeutralElement(int,IntegerMultiplication)
    by (rule monoid0.group0_1_L4)
  then show  ($# 1) = 1 by simp
qed
```

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

```
lemma (in int0) Int_ZF_1_L8A: shows 0 ∈ ℤ  1 ∈ ℤ
proof -
  have ($# 0) ∈ ℤ  ($# 1) ∈ ℤ by auto
  then show 0 ∈ ℤ  1 ∈ ℤ using Int_ZF_1_L8 by auto
qed
```

Zero is not one.

```
lemma (in int0) int_zero_not_one: shows 0 ≠ 1
proof -
  have ($# 0) ≠ ($# 1) by simp
  then show 0 ≠ 1 using Int_ZF_1_L8 by simp
qed
```

The set of integers is not empty, of course.

```
lemma (in int0) int_not_empty: shows ℤ ≠ 0
  using Int_ZF_1_L8A by auto
```

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

```
lemma (in int0) int_not_trivial: shows ℤ ≠ {0}
  using Int_ZF_1_L8A int_zero_not_one by blast
```

Each integer has an inverse (in the addition sense).

```
lemma (in int0) Int_ZF_1_L9: assumes A1: g ∈ ℤ
  shows ∃ b∈ℤ. g+b = 0
proof -
  from A1 have g+ $-g = 0
    using Int_ZF_1_L2 Int_ZF_1_L8 by simp
  thus thesis by auto
qed
```

Integers with addition form an abelian group. This also shows that we can apply all theorems proven in the proof contexts (locales) that require the assumpion that some pair of sets form a group like locale group0.

```
theorem Int_ZF_1_T2: shows
  IsAgroup(int,IntegerAddition)
  IntegerAddition {is commutative on} int
  group0(int,IntegerAddition)
  using int0.Int_ZF_1_T1 int0.Int_ZF_1_L9 IsAgroup_def
  group0_def int0.Int_ZF_1_L4 IsCommutative_def by auto
```

What is the additive group inverse in the group of integers?

```
lemma (in int0) Int_ZF_1_L9A: assumes A1: m∈ℤ
  shows $-m = -m
proof -
   from A1 have m∈int $-m ∈ int IntegerAddition<m,$-m> =
     TheNeutralElement(int,IntegerAddition)
    using zminus_type Int_ZF_1_L2 Int_ZF_1_L8 by auto
  then have $-m = GroupInv(int,IntegerAddition)(m)
    using Int_ZF_1_T2 group0.group0_2_L9 by blast
  then show thesis by simp
qed
```

Subtracting integers corresponds to adding the negative.

```
lemma (in int0) Int_ZF_1_L10: assumes A1: m∈ℤ  n∈ℤ
  shows m-n = m $+ $-n
  using prems Int_ZF_1_T2  group0.inverse_in_group Int_ZF_1_L9A Int_ZF_1_L2
  by simp
```

Negative of zero is zero.

```
lemma (in int0) Int_ZF_1_L11: shows (-0) = 0
  using Int_ZF_1_T2  group0.group_inv_of_one by simp
```

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

```
lemma Int_ZF_1_L12:
  assumes m∈int shows m $- $#1 $+ $#1 = m
  using prems eq_zdiff_iff by auto
```

A trivial calculation lemma that allows to subtract and add one, version with ZF-operation.

```
lemma (in int0) Int_ZF_1_L13: assumes m∈ℤ
  shows (m $- $#1) + 1 = m
  using prems Int_ZF_1_L8A Int_ZF_1_L2 Int_ZF_1_L8 Int_ZF_1_L12
  by simp
```

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 -
  { assume m+1 = m
    with A1 have
      group0(ℤ,IntegerAddition)
      m∈ℤ  1∈ℤ
      IntegerAddition⟨m,1⟩ = m
      using Int_ZF_1_T2 Int_ZF_1_L8A by auto
    then have 1 = TheNeutralElement(ℤ,IntegerAddition)
      by (rule group0.group0_2_L7)
    then have False using int_zero_not_one by simp
  } then show I: m+1 ≠ m by auto
  { from A1 have m - 1 + 1 = m
      using Int_ZF_1_L8A Int_ZF_1_T2 group0.group0_2_L16
      by simp
    moreover assume m-1 = m
    ultimately have m + 1 = m by simp
    with I have False by simp
  } then show m-1 ≠ m by auto
qed
```

If the difference is zero, the integers are equal.

```
lemma (in int0) Int_ZF_1_L15:
  assumes A1: m∈ℤ  n∈ℤ and A2: m-n = 0
  shows m=n
proof -
  let G = ℤ
  let f = IntegerAddition
  from A1 A2 have
    group0(G, f)
    m ∈ G  n ∈ G
    f⟨m, GroupInv(G, f)(n)⟩ = TheNeutralElement(G, f)
    using Int_ZF_1_T2 by auto
  then show m=n by (rule group0.group0_2_L11A)
qed
```

23.2 Integers as an ordered group

In this section we define order on integers as a relation, that is a subset of $Z \times Z$ and show that integers form an ordered group.

The next lemma interprets the order definition one way.

```
lemma (in int0) Int_ZF_2_L1:
  assumes A1: m∈ℤ n∈ℤ and A2: m $≤ n
  shows m ≤ n
proof -
  from A1 A2 have <m,n> ∈ {x∈ℤ×ℤ. fst(x) $≤ snd(x)}
    by simp
  then show thesis using IntegerOrder_def by simp
```

```
qed
```

The next lemma interprets the definition the other way.

```
lemma (in int0) Int_ZF_2_L1A: assumes A1: m ≤ n
  shows m $≤ n m∈ℤ n∈ℤ
proof -
  from A1 have <m,n> ∈ {p∈ℤ×ℤ. fst(p) $≤ snd(p)}
    using IntegerOrder_def by simp
  thus m $≤ n  m∈ℤ  n∈ℤ by auto
qed
```

Integer order is a relation on integers.

```
lemma Int_ZF_2_L1B: IntegerOrder ⊆ int×int
proof
  fix x assume x∈IntegerOrder
  then have x ∈ {p∈int×int. fst(p) $≤ snd(p)}
    using IntegerOrder_def by simp
  then show x∈int×int by simp
qed
```

The way we define the notion of being bounded below, its sufficient for the relation to be on integers for all bounded below sets to be subsets of integers.

```
lemma (in int0) Int_ZF_2_L1C:
  assumes A1: IsBoundedBelow(A,IntegerOrder)
  shows A⊆ℤ
proof -
  from A1 have
    IntegerOrder ⊆ ℤ×ℤ
    IsBoundedBelow(A,IntegerOrder)
    using Int_ZF_2_L1B by auto
  then show A⊆ℤ by (rule Order_ZF_3_L1B)
qed
```

The order on integers is reflexive.

```
lemma (in int0) int_ord_is_refl: shows refl(ℤ,IntegerOrder)
  using Int_ZF_2_L1 zle_refl refl_def by auto
```

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

```
lemma (in int0) Int_ZF_2_L3:
  assumes A1: m ≤ n  n ≤ m
  shows m=n
proof -
  from A1 have m $≤ n  n $≤ m  m∈ℤ  n∈ℤ
    using Int_ZF_2_L1A by auto
  then show m=n using zle_anti_sym by auto
qed
```

The order on integers is antisymmetric.

```
lemma (in int0) Int_ZF_2_L4: antisym(IntegerOrder)
proof -
  have ∀m n. m ≤ n  ∧ n ≤ m ⟶ m=n
    using Int_ZF_2_L3 by auto
  then show thesis using imp_conj antisym_def by simp
qed
```

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

```
lemma Int_ZF_2_L5:
  assumes A1: ⟨m,n⟩ ∈ IntegerOrder  ⟨n,k⟩ ∈ IntegerOrder
  shows ⟨m,k⟩ ∈ IntegerOrder
proof -
  from A1 have T1: m $≤ n n $≤ k and T2: m∈int k∈int
    using int0.Int_ZF_2_L1A by auto
  from T1 have m $≤ k by (rule zle_trans)
  with T2 show thesis using int0.Int_ZF_2_L1 by simp
qed
```

The order on integers is transitive. This version is stated in the `int0` context using notation for integers.

```
lemma (in int0) Int_order_transitive:
  assumes A1: m≤n  n≤k
  shows m≤k
proof -
  from A1 have <m,n> ∈ IntegerOrder  <n,k> ∈ IntegerOrder
    by auto
  then have <m,k> ∈ IntegerOrder by (rule Int_ZF_2_L5)
  then show m≤k by simp
qed
```

The order on integers is transitive.

```
lemma Int_ZF_2_L6: trans(IntegerOrder)
proof -
  have ∀ m n k.
    ⟨m, n⟩ ∈ IntegerOrder ∧ ⟨n, k⟩ ∈ IntegerOrder ⟶
    ⟨m, k⟩ ∈ IntegerOrder
    using Int_ZF_2_L5 by blast
  then show thesis by (rule Fol1_L2)
qed
```

The order on integers is a partial order.

```
lemma Int_ZF_2_L7: shows IsPartOrder(int,IntegerOrder)
  using int0.int_ord_is_refl int0.Int_ZF_2_L4
    Int_ZF_2_L6 IsPartOrder_def by simp
```

The essential condition to show that the order on integers is preserved by translations.

```
lemma (in int0) int_ord_transl_inv:
```

```
  assumes A1: k ∈ ℤ and A2: m ≤ n
  shows m+k ≤ n+k   k+m≤ k+n
proof -
  from A2 have m $≤ n and m∈ℤ n∈ℤ
    using Int_ZF_2_L1A by auto
  with A1 show m+k ≤ n+k   k+m≤ k+n
    using zadd_right_cancel_zle zadd_left_cancel_zle
    Int_ZF_1_L2 Int_ZF_1_L1 apply_funtype
    Int_ZF_1_L2 Int_ZF_2_L1 Int_ZF_1_L2 by auto
qed
```

Integers form a linearly ordered group. We can apply all theorems proven in group3 context to integers.

```
theorem (in int0) Int_ZF_2_T1: shows
  IsAnOrdGroup(ℤ,IntegerAddition,IntegerOrder)
  IntegerOrder {is total on} ℤ
  group3(ℤ,IntegerAddition,IntegerOrder)
  IsLinOrder(ℤ,IntegerOrder)
proof -
  have ∀k∈ℤ. ∀m n. m ≤ n  ⟶
    m+k ≤ n+k ∧ k+m≤ k+n
    using int_ord_transl_inv by simp
  then show T1: IsAnOrdGroup(ℤ,IntegerAddition,IntegerOrder) using
    Int_ZF_1_T2 Int_ZF_2_L1B Int_ZF_2_L7 IsAnOrdGroup_def
    by simp
  then show group3(ℤ,IntegerAddition,IntegerOrder)
    using group3_def by simp
  show IntegerOrder {is total on} ℤ
    using IsTotal_def zle_linear Int_ZF_2_L1 by auto
  with T1 show IsLinOrder(ℤ,IntegerOrder)
    using IsAnOrdGroup_def IsPartOrder_def IsLinOrder_def by simp
qed
```

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 -
  from A1 have i $≤ m  i∈ℤ  m∈ℤ
    using Int_ZF_2_L1A by auto
  with A2 show i $< m using zle_def by simp
qed
```

This shows how Isabelle's `$<` operator translates to IsarMathLib notation.

```
lemma (in int0) Int_ZF_2_L9AA: assumes A1: m∈ℤ  n∈ℤ
  and A2: m $< n
  shows m≤n  m ≠ n
  using prems zle_def Int_ZF_2_L1 by auto
```

A small technical lemma about putting one on the other side of an inequality.

```
lemma (in int0) Int_ZF_2_L9A:
  assumes A1: k∈ℤ and A2: m ≤ k $- ($# 1)
  shows m+1 ≤ k
proof -
  from A2 have m+1 ≤ (k $- ($# 1)) + 1
    using Int_ZF_1_L8A int_ord_transl_inv by simp
  with A1 show m+1 ≤ k
    using Int_ZF_1_L13 by simp
qed
```

We can put any integer on the other side of an inequality reversing its sign.

```
lemma (in int0) Int_ZF_2_L9B: assumes i∈ℤ  m∈ℤ  k∈ℤ
  shows i+m ≤ k  ⟷  i ≤ k-m
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L9A
  by simp
```

A special case of `Int_ZF_2_L9B` with weaker assumptions.

```
lemma (in int0) Int_ZF_2_L9C:
  assumes i∈ℤ  m∈ℤ and i-m ≤ k
  shows i ≤ k+m
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L9B
  by simp
```

Taking (higher order) minus on both sides of inequality reverses it.

```
lemma (in int0) Int_ZF_2_L10: assumes k ≤ i
  shows
  (-i) ≤ (-k)
  $-i ≤ $-k
  using prems Int_ZF_2_L1A Int_ZF_1_L9A Int_ZF_2_T1
    group3.OrderedGroup_ZF_1_L5 by auto
```

Taking minus on both sides of inequality reverses it, version with a negative on one side.

```
lemma (in int0) Int_ZF_2_L10AA: assumes n∈ℤ  m≤(-n)
  shows n≤(-m)
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5AD
  by simp
```

We can cancel the same element on on both sides of an inequality, a version with minus on both sides.

```
lemma (in int0) Int_ZF_2_L10AB:
  assumes m∈ℤ  n∈ℤ  k∈ℤ and m-n ≤ m-k
  shows k≤n
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5AF
  by simp
```

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

```
lemma (in int0) Int_ZF_2_L10A: assumes k ≤ 0
  shows 0≤(-k)
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5A by simp
```

If the opposite of an integers is nonnegative, then the integer is nonpositive.

```
lemma (in int0) Int_ZF_2_L10B:
  assumes k∈ℤ and 0≤(-k)
  shows k≤0
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5AA by simp
```

Adding one to an integer corresponds to taking a successor for a natural number.

```
lemma (in int0) Int_ZF_2_L11: i $+ $# n $+ ($# 1)  =  i $+ $# succ(n)
proof -
  have $# succ(n) = $#1 $+ $# n using int_succ_int_1 by blast
  then have i $+ $# succ(n) = i $+ ($# n  $+ $#1)
    using zadd_commute by simp
  then show thesis using zadd_assoc by simp
qed
```

Adding a natural number increases integers.

```
lemma (in int0) Int_ZF_2_L12: assumes A1: i∈ℤ and A2: n∈nat
  shows i ≤ i $+ $#n
proof (cases n = 0)
  assume n = 0
  with A1 show i ≤ i $+ $#n using zadd_int0 int_ord_is_refl refl_def
    by simp
next
  assume n≠0
  with A2 obtain k where k∈nat n = succ(k)
    using Nat_ZF_1_L3 by auto
  with A1 show i ≤ i $+ $#n
    using zless_succ_zadd zless_imp_zle Int_ZF_2_L1 by simp
qed
```

Adding one increases integers.

```
lemma (in int0) Int_ZF_2_L12A: assumes A1: j≤k
  shows j ≤ k $+ $#1  j ≤ k+1
proof -
  from A1 have T1:j∈ℤ k∈ℤ j $≤ k
    using Int_ZF_2_L1A by auto
  moreover from T1 have k $≤ k $+ $#1 using Int_ZF_2_L12 Int_ZF_2_L1A
    by simp
  ultimately have j $≤ k $+ $#1 using zle_trans by fast
  with T1 show j ≤ k $+ $#1 using Int_ZF_2_L1 by simp
  with T1 have j≤ k+$#1
    using Int_ZF_1_L2 by simp
  then show j ≤ k+1 using Int_ZF_1_L8 by simp
```

```
qed
```

Adding one increases integers, yet one more version.

```
lemma (in int0) Int_ZF_2_L12B: assumes A1: m∈ℤ shows m ≤ m+1
  using prems int_ord_is_refl refl_def Int_ZF_2_L12A by simp
```

If $k + 1 = m + n$, where n is a non-zero natural number, then $m \leq k$.

```
lemma (in int0) Int_ZF_2_L13:
  assumes A1: k∈ℤ m∈ℤ and A2: n∈nat
  and A3: k $+ ($# 1) = m $+ $# succ(n)
  shows m ≤ k
proof -
  from A1 have k∈ℤ m $+ $# n ∈ ℤ by auto
  moreover from A2 have k $+ $# 1 = m $+ $# n $+ $#1
    using Int_ZF_2_L11 by simp
  ultimately have k = m $+ $# n using zadd_right_cancel by simp
  with A1 A2 show thesis using Int_ZF_2_L12 by simp
qed
```

The absolute value of an integer is an integer.

```
lemma (in int0) Int_ZF_2_L14: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ
proof -
  have AbsoluteValue(ℤ,IntegerAddition,IntegerOrder) : ℤ→ℤ
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L1 by simp
  with A1 show thesis using apply_funtype by simp
qed
```

If two integers are nonnegative, then the opposite of one is less or equal than the other and the sum is also nonnegative.

```
lemma (in int0) Int_ZF_2_L14A:
  assumes 0≤m  0≤n
  shows
  (-m) ≤ n
  0 ≤ m + n
  using prems Int_ZF_2_T1
    group3.OrderedGroup_ZF_1_L5AC group3.OrderedGroup_ZF_1_L12
  by auto
```

We can increase components in an estimate.

```
lemma (in int0) Int_ZF_2_L15:
  assumes b≤b1 c≤c1 and a≤b+c
  shows a≤b1+c1
proof -
  from prems have group3(ℤ,IntegerAddition,IntegerOrder)
    ⟨a,IntegerAddition<b,c>⟩ ∈ IntegerOrder
    ⟨b,b1⟩ ∈ IntegerOrder ⟨c,c1⟩ ∈ IntegerOrder
    using Int_ZF_2_T1 by auto
```

```
  then have ⟨a,IntegerAddition<b1,c1>⟩ ∈ IntegerOrder
    by (rule group3.OrderedGroup_ZF_1_L5E)
  thus thesis by simp
qed
```

We can add or subtract the sides of two inequalities.

```
lemma (in int0) int_ineq_add_sides:
  assumes a≤b and c≤d
  shows
  a+c ≤ b+d
  a-d ≤ b-c
  using prems Int_ZF_2_T1
    group3.OrderedGroup_ZF_1_L5B group3.OrderedGroup_ZF_1_L5I
  by auto
```

We can increase the second component in an estimate.

```
lemma (in int0) Int_ZF_2_L15A:
  assumes b∈ℤ and a≤b+c and A3: c≤c1
  shows a≤b+c1
proof -
  from prems have
    group3(ℤ,IntegerAddition,IntegerOrder)
    b ∈ ℤ
    ⟨a,IntegerAddition<b,c>⟩ ∈ IntegerOrder
    ⟨c,c1⟩ ∈ IntegerOrder
    using Int_ZF_2_T1 by auto
  then have ⟨a,IntegerAddition<b,c1>⟩ ∈ IntegerOrder
     by (rule group3.OrderedGroup_ZF_1_L5D)
   thus thesis by simp
qed
```

If we increase the second component in a sum of three integers, the whole sum inceases.

```
lemma (in int0) Int_ZF_2_L15C:
  assumes A1: m∈ℤ  n∈ℤ and A2: k ≤ L
  shows m+k+n ≤ m+L+n
proof -
  let P = IntegerAddition
  from prems have
    group3(int,P,IntegerOrder)
    m ∈ int  n ∈ int
    ⟨k,L⟩ ∈ IntegerOrder
    using Int_ZF_2_T1 by auto
  then have ⟨P⟨P<m,k>,n⟩, P⟨P<m,L>,n⟩ ⟩ ∈ IntegerOrder
    by (rule group3.OrderedGroup_ZF_1_L10)
  then show m+k+n ≤ m+L+n by simp
qed
```

We don't decrease an integer by adding a nonnegative one.

```
lemma (in int0) Int_ZF_2_L15D:
  assumes 0≤n  m∈ℤ
  shows m ≤ n+m
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5F
  by simp
```

Some inequalities about the sum of two integers and its absolute value.

```
lemma (in int0) Int_ZF_2_L15E:
  assumes m∈ℤ  n∈ℤ
  shows
  m+n ≤ abs(m)+abs(n)
  m-n ≤ abs(m)+abs(n)
  (-m)+n ≤ abs(m)+abs(n)
  (-m)-n ≤ abs(m)+abs(n)
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L6A
  by auto
```

We can add a nonnegative integer to the right hand side of an inequality.

```
lemma (in int0) Int_ZF_2_L15F:  assumes m≤k  and 0≤n
  shows m ≤ k+n  m ≤ n+k
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L5G
  by auto
```

Triangle inequality for integers.

```
lemma (in int0) Int_triangle_ineq:
  assumes m∈ℤ  n∈ℤ
  shows abs(m+n)≤abs(m)+abs(n)
  using prems Int_ZF_1_T2 Int_ZF_2_T1 group3.OrdGroup_triangle_ineq
  by simp
```

Taking absolute value does not change nonnegative integers.

```
lemma (in int0) Int_ZF_2_L16:
  assumes 0≤m shows  m∈ℤ⁺ and abs(m) = m
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L2
    group3.OrderedGroup_ZF_3_L2 by auto
```

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

```
lemma (in int0) Int_ZF_2_L16A: shows 0≤1 and abs(1) = 1
proof -
  have ($# 0) ∈ ℤ ($# 1)∈ ℤ by auto
  then have 0≤0 and T1: 1∈ℤ
    using Int_ZF_1_L8 int_ord_is_refl refl_def by auto
  then have 0≤0+1 using Int_ZF_2_L12A by simp
  with T1 show 0≤1 using Int_ZF_1_T2 group0.group0_2_L2
    by simp
  then show abs(1) = 1 using Int_ZF_2_L16 by simp
qed
```

$1 \leq 2$.

```
lemma (in int0) Int_ZF_2_L16B: shows 1≤2
proof -
  have ($# 1)∈ ℤ by simp
  then show 1≤2
    using Int_ZF_1_L8 int_ord_is_refl refl_def Int_ZF_2_L12A
    by simp
qed
```

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 -
  from A1 have 0≤1 and 1≤a
    using Int_ZF_2_L16A by auto
  then show 0≤a by (rule Int_order_transitive)
  have I: 0≤1 using Int_ZF_2_L16A by simp
  have 1≤2 using Int_ZF_2_L16B by simp
  moreover from A1 show 2 ≤ a+1
    using Int_ZF_1_L8A int_ord_transl_inv by simp
  ultimately show 1 ≤ a+1 by (rule Int_order_transitive)
  with I show 0 ≤ a+1 by (rule Int_order_transitive)
  from A1 show a≠0 using
    Int_ZF_2_L16A Int_ZF_2_L3 int_zero_not_one by auto
qed
```

Absolute value is the same for an integer and its opposite.

```
lemma (in int0) Int_ZF_2_L17:
  assumes m∈ℤ shows abs(-m) = abs(m)
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L7A by simp
```

The absolute value of zero is zero.

```
lemma (in int0) Int_ZF_2_L18: shows abs(0) = 0
  using Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L2A by simp
```

A different version of the triangle inequality.

```
lemma (in int0) Int_triangle_ineq1:
  assumes A1: m∈ℤ  n∈ℤ
  shows
  abs(m-n) ≤ abs(n)+abs(m)
  abs(m-n) ≤ abs(m)+abs(n)
proof -
  have $-n ∈ ℤ by simp
  with A1 have abs(m-n) ≤ abs(m)+abs(-n)
    using Int_ZF_1_L9A Int_triangle_ineq by simp
```

```
  with A1 show
    abs(m-n) ≤ abs(n)+abs(m)
    abs(m-n) ≤ abs(m)+abs(n)
    using Int_ZF_2_L17 Int_ZF_2_L14 Int_ZF_1_T2 IsCommutative_def
    by auto
qed
```

Another version of the triangle inequality.

```
lemma (in int0) Int_triangle_ineq2:
  assumes m∈ℤ  n∈ℤ
  and abs(m-n) ≤ k
  shows
  abs(m) ≤ abs(n)+k
  m-k ≤ n
  m ≤ n+k
  n-k ≤ m
  using prems Int_ZF_1_T2 Int_ZF_2_T1
    group3.OrderedGroup_ZF_3_L7D group3.OrderedGroup_ZF_3_L7E
  by auto
```

Triangle inequality with three integers. We could use `OrdGroup_triangle_ineq3`, but since simp cannot translate the notation directly, it is simpler to reprove it for integers.

```
lemma (in int0) Int_triangle_ineq3:
  assumes A1: m∈ℤ  n∈ℤ  k∈ℤ
  shows abs(m+n+k) ≤ abs(m)+abs(n)+abs(k)
proof -
  from A1 have T: m+n ∈ ℤ  abs(k) ∈ ℤ
    using Int_ZF_1_T2 group0.group_op_closed  Int_ZF_2_L14
    by auto
  with A1 have abs(m+n+k) ≤ abs(m+n) + abs(k)
    using Int_triangle_ineq by simp
  moreover from A1 T have
    abs(m+n) + abs(k) ≤ abs(m) + abs(n) + abs(k)
    using Int_triangle_ineq int_ord_transl_inv by simp
  ultimately show thesis by (rule Int_order_transitive)
qed
```

The next lemma shows what happens when one integers is not greater or equal than another.

```
lemma (in int0) Int_ZF_2_L19:
  assumes A1: m∈ℤ  n∈ℤ and A2: ¬(n≤m)
  shows m≤n  (-n) ≤ (-m)  m≠n
proof -
  from A1 A2 show m≤n using Int_ZF_2_T1 IsTotal_def
    by auto
  then show (-n) ≤ (-m) using Int_ZF_2_L10
    by simp
```

```
  from A1 have n ≤ n using int_ord_is_refl refl_def
    by simp
  with A2 show m≠n by auto
qed
```

If one integer is greater or equal and not equal to another, then it is not smaller or equal.

```
lemma (in int0) Int_ZF_2_L19AA: assumes A1: m≤n and A2: m≠n
  shows ¬(n≤m)
proof -
  from A1 A2 have
    group3(ℤ, IntegerAddition, IntegerOrder)
    ⟨m,n⟩ ∈ IntegerOrder
    m≠n
    using Int_ZF_2_T1 by auto
  then have ⟨n,m⟩ ∉ IntegerOrder
    by (rule group3.OrderedGroup_ZF_1_L8AA)
  thus ¬(n≤m) by simp
qed
```

The next lemma allows to prove theorems for the case of positive and negative integers separately.

```
lemma (in int0) Int_ZF_2_L19A: assumes A1: m∈ℤ and A2: ¬(0≤m)
  shows m≤0  0 ≤ (-m)  m≠0
proof -
  from A1 have T1: 0 ∈ ℤ
    using Int_ZF_1_T2 group0.group0_2_L2 by auto
  with A1 show m≤0 by (rule Int_ZF_2_L19)
  from A1 T1 show m≠0  by (rule Int_ZF_2_L19)
  from A1 T1 have (-0)≤(-m) by (rule Int_ZF_2_L19)
  then show 0 ≤ (-m)
    using Int_ZF_1_T2 group0.group_inv_of_one by simp
qed
```

We can prove a theorem about integers by proving that it holds for $m = 0$, $m \in\mathbb{Z}_+$ and $-m \in\mathbb{Z}_+$.

```
lemma (in int0) Int_ZF_2_L19B:
  assumes m∈ℤ and Q(0) and ∀n∈ℤ₊. Q(n) and ∀n∈ℤ₊. Q(-n)
  shows Q(m)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  let b = m
  from prems have
    group3(G, P, r)
    r {is total on} G
    b ∈ G
```

```
    Q(TheNeutralElement(G, P))
    ∀a∈PositiveSet(G, P, r). Q(a)
    ∀a∈PositiveSet(G, P, r). Q(GroupInv(G, P)(a))
    using Int_ZF_2_T1 by auto
  then show Q(b) by (rule group3.OrderedGroup_ZF_1_L18)
qed
```

An integer is not greater than its absolute value.

```
lemma (in int0) Int_ZF_2_L19C: assumes A1: m∈ℤ
  shows
  m ≤ abs(m)
  (-m) ≤ abs(m)
  using prems Int_ZF_2_T1
    group3.OrderedGroup_ZF_3_L5 group3.OrderedGroup_ZF_3_L6
  by auto
```

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

```
lemma (in int0) Int_ZF_2_L20: assumes m∈ℤ  n∈ℤ
  shows abs(m-n) = abs(n-m)
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L7B by simp
```

We can add the sides of inequalities with absolute values.

```
lemma (in int0) Int_ZF_2_L21:
  assumes A1: m∈ℤ n∈ℤ
  and A2: abs(m) ≤ k  abs(n) ≤ l
  shows
  abs(m+n) ≤ k + l
  abs(m-n) ≤ k + l
  using prems Int_ZF_1_T2 Int_ZF_2_T1
    group3.OrderedGroup_ZF_3_L7C group3.OrderedGroup_ZF_3_L7CA
  by auto
```

Absolute value is nonnegative.

```
lemma (in int0) int_abs_nonneg: assumes A1: m∈ℤ
  shows abs(m) ∈ ℤ⁺  0 ≤ abs(m)
proof -
  have AbsoluteValue(ℤ,IntegerAddition,IntegerOrder) : ℤ→ℤ⁺
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L3C by simp
  with A1 show abs(m) ∈ ℤ⁺ using apply_funtype
    by simp
  then show 0 ≤ abs(m)
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L2 by simp
qed
```

If an nonnegative integer is less or equal than another, then so is its absolute value.

```
lemma (in int0) Int_ZF_2_L23:
  assumes 0≤m   m≤k
```

```
  shows abs(m) ≤ k
  using prems Int_ZF_2_L16 by simp
```

23.3 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 (cases i=m)
  let n = 0
  assume A2: i=m
  from A1 A2 have n ∈ nat m = i $+ $# n
    using Int_ZF_2_L1A zadd_int0_right by auto
  thus ∃n∈nat. m = i $+ $# n by blast
next
  assume A3: i≠m
  with A1 have i $< m i∈ℤ m∈ℤ
    using Int_ZF_2_L9 Int_ZF_2_L1A by auto
  then obtain k where D1: k∈nat m = i $+ $# succ(k)
    using zless_imp_succ_zadd_lemma by auto
  let n = succ(k)
  from D1 have n∈nat m = i $+ $# n by auto
  thus ∃n∈nat. m = i $+ $# n by simp
qed
```

Induction for integers, the induction step.

```
lemma (in int0) Int_ZF_3_L6: assumes A1: i∈ℤ
  and A2: ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
  shows ∀k∈nat. Q(i $+ ($# k)) ⟶ Q(i $+ ($# succ(k)))
proof
  fix k assume A3: k∈nat show Q(i $+ $# k) ⟶ Q(i $+ $# succ(k))
  proof
    assume A4: Q(i $+ $# k)
    from A1 A3 have i≤ i $+ ($# k) using Int_ZF_2_L12
      by simp
    with A4 A2 have Q(i $+ ($# k) $+ ($# 1)) by simp
    then show Q(i $+ ($# succ(k))) using Int_ZF_2_L11 by simp
  qed
qed
```

Induction on integers, version with higher-order increment function.

```
lemma (in int0) Int_ZF_3_L7:
  assumes A1: i≤k and A2: Q(i)
  and A3: ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
```

```
  shows Q(k)
proof -
  from A1 obtain n where D1: n∈nat and D2: k = i $+ $# n
    using Int_ZF_3_L2 by auto
  from A1 have T1: i∈ℤ using Int_ZF_2_L1A by simp
  from D1 have n∈nat .
  moreover from A1 have Q(i $+ $#0)
    using Int_ZF_2_L1A zadd_int0 by simp
  moreover from T1 A3 have
    ∀k∈nat. Q(i $+ ($# k)) ⟶ Q(i $+ ($# succ(k)))
    by (rule Int_ZF_3_L6)
  ultimately have Q(i $+ ($# n)) by (rule Nat_ZF_1_L2)
  with D2 show Q(k) by simp
qed
```

Induction on integer, implication between two forms of the induction step.

```
lemma (in int0) Int_ZF_3_L7A: assumes
  A1: ∀m. i≤m ∧ Q(m) ⟶ Q(m+1)
  shows ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
proof -
  { fix m assume i≤m ∧ Q(m)
    with A1 have T1: m∈ℤ Q(m+1) using Int_ZF_2_L1A by auto
    then have m+1 = m+($# 1) using Int_ZF_1_L8 by simp
    with T1 have Q(m $+ ($# 1)) using Int_ZF_1_L2
      by simp
  } then show thesis by simp
qed
```

Induction on integers, version with ZF increment function.

```
theorem (in int0) Induction_on_int:
  assumes A1: i≤k and A2: Q(i)
  and A3: ∀m. i≤m ∧ Q(m) ⟶ Q(m+1)
  shows Q(k)
proof -
  from A3 have ∀m. i≤m ∧ Q(m) ⟶ Q(m $+ ($# 1))
    by (rule Int_ZF_3_L7A)
  with A1 A2 show thesis by (rule Int_ZF_3_L7)
qed
```

Another form of induction on integers. This rewrites the basic theorem Int_ZF_3_L7 substituting $P(-k)$ for $Q(k)$.

```
lemma (in int0) Int_ZF_3_L7B: assumes A1: i≤k and A2: P($-i)
  and A3: ∀m. i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  shows P($-k)
proof -
  from A1 A2 A3 show P($-k) by (rule Int_ZF_3_L7)
qed
```

Another induction on integers. This rewrites Int_ZF_3_L7 substituting $-k$

for k and $-i$ for i.

```
lemma (in int0) Int_ZF_3_L8: assumes A1: k≤i and A2: P(i)
  and A3: ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  shows P(k)
proof -
  from A1 have T1: $-i≤$-k using Int_ZF_2_L10 by simp
  from A1 A2 have T2: P($- $- i) using Int_ZF_2_L1A zminus_zminus
    by simp
  from T1 T2 A3 have P($-($-k)) by (rule Int_ZF_3_L7)
  with A1 show P(k) using Int_ZF_2_L1A zminus_zminus by simp
qed
```

An implication between two forms of induction steps.

```
lemma (in int0) Int_ZF_3_L9: assumes A1: i∈ℤ
  and A2: ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
  shows ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
proof
  fix m show $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
  proof
    assume A3: $- i ≤ m ∧ P($- m)
    then have $- i ≤ m by simp
    then have $-m ≤ $- ($- i) by (rule Int_ZF_2_L10)
    with A1 A2 A3 show P($-(m $+ ($# 1)))
      using zminus_zminus zminus_zadd_distrib by simp
  qed
qed
```

Backwards induction on integers, version with higher-order decrement function.

```
lemma (in int0) Int_ZF_3_L9A: assumes A1: k≤i and A2: P(i)
  and A3: ∀n. n≤i ∧ P(n) ⟶P(n $+ $-($#1))
  shows P(k)
proof -
  from A1 have T1: i∈ℤ using Int_ZF_2_L1A by simp
  from T1 A3 have T2: ∀m. $-i≤m ∧ P($-m) ⟶ P($-(m $+ ($# 1)))
    by (rule Int_ZF_3_L9)
  from A1 A2 T2 show P(k) by (rule Int_ZF_3_L8)
qed
```

Induction on integers, implication between two forms of the induction step.

```
lemma (in int0) Int_ZF_3_L10: assumes
  A1: ∀n. n≤i ∧ P(n) ⟶ P(n-1)
  shows ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
proof -
  { fix n assume n≤i ∧ P(n)
    with A1 have T1: n∈ℤ P(n-1) using Int_ZF_2_L1A by auto
    then have n-1 = n-($# 1) using Int_ZF_1_L8 by simp
    with T1 have P(n $+ $-($#1)) using Int_ZF_1_L10 by simp
```

```
  } then show thesis by simp
qed
```

Backwards induction on integers.

```
theorem (in int0) Back_induct_on_int:
  assumes A1: k≤i and A2: P(i)
  and A3: ∀n. n≤i ∧ P(n) ⟶ P(n-1)
  shows P(k)
proof -
  from A3 have ∀n. n≤i ∧ P(n) ⟶ P(n $+ $-($#1))
    by (rule Int_ZF_3_L10)
  with A1 A2 show P(k) by (rule Int_ZF_3_L9A)
qed
```

23.4 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 (cases n=0)
  assume n=0
  with A1 A2 show m =  k $+ $#1 ∨ m ≤ k
    using zadd_int0 by simp
next assume n≠0
  with A1 obtain j where D1: j∈nat n = succ(j)
    using Nat_ZF_1_L3 by auto
  with A1 A2 D1 show m =  k $+ $#1 ∨ m ≤ k
    using Int_ZF_2_L13 by simp
qed
```

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

```
lemma Int_ZF_4_L1A:
  assumes m∈int shows m $- $#1 $+ $#1 = m
  using prems eq_zdiff_iff by auto
```

There are no integers between k and $k+1$, another formulation.

```
lemma (in int0) Int_ZF_4_L1B: assumes A1: m ≤ L
  shows
  m = L ∨ m+1 ≤ L
  m = L ∨ m ≤ L-1
proof -
```

```
  let k = L $- $#1
  from A1 have T1: m∈ℤ  L∈ℤ  L = k $+ $#1
    using Int_ZF_2_L1A Int_ZF_4_L1A by auto
  moreover from A1 obtain n where D1: n∈nat  L = m $+ $# n
    using Int_ZF_3_L2 by auto
  ultimately have m = L ∨ m ≤ k
    using Int_ZF_4_L1 by simp
  with T1 show m = L   ∨  m+1 ≤ L
    using Int_ZF_2_L9A by auto
  with T1 show m = L ∨ m ≤ L-1
    using Int_ZF_1_L8A Int_ZF_2_L9B by simp
qed
```

If $j \in m..k+1$, then $j \in m..n$ or $j = k+1$.

```
lemma (in int0) Int_ZF_4_L2: assumes A1: k∈ℤ
  and A2: j ∈ m..(k $+ $#1)
  shows j ∈ m..k ∨ j ∈ {k $+ $#1}
proof -
  from A2 have T1: m≤j j≤(k $+ $#1) using Order_ZF_2_L1A
    by auto
  then have T2: m∈ℤ j∈ℤ using Int_ZF_2_L1A by auto
  from T1 obtain n where n∈nat k $+ $#1 = j $+ $# n
    using Int_ZF_3_L2 by auto
  with A1 T1 T2 have (m≤j ∧ j ≤ k) ∨ j ∈ {k $+ $#1}
    using Int_ZF_4_L1 by auto
  then show thesis using Order_ZF_2_L1B by auto
qed
```

Extending an integer interval by one is the same as adding the new endpoint.

```
lemma (in int0) Int_ZF_4_L3: assumes A1: m≤ k
  shows m..(k $+ $#1) = m..k ∪ {k $+ $#1}
proof
  from A1 have T1: m∈ℤ k∈ℤ using Int_ZF_2_L1A by auto
  then show m .. (k $+ $# 1) ⊆ m .. k ∪ {k $+ $# 1}
    using Int_ZF_4_L2 by auto
  from T1 have m≤ m using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L3
    by simp
  with T1 A1 have m .. k ⊆ m .. (k $+ $# 1)
    using Int_ZF_2_L12 Int_ZF_2_L6 Order_ZF_2_L3 by simp
  with T1 A1 show m..k ∪ {k $+ $#1} ⊆ m..(k $+ $#1)
    using Int_ZF_2_L12A int_ord_is_refl Order_ZF_2_L2 by auto
qed
```

Integer intervals are finite - induction step.

```
lemma (in int0) Int_ZF_4_L4:
  assumes A1: i≤m and A2: i..m ∈ Fin(ℤ)
  shows i..(m $+ $#1) ∈ Fin(ℤ)
  using prems Int_ZF_4_L3 by simp
```

Integer intervals are finite.

```
lemma (in int0) Int_ZF_4_L5: assumes A1: i∈ℤ k∈ℤ
  shows i..k ∈ Fin(ℤ)
proof (cases i≤ k)
  assume A2: i≤k
  moreover from A1 have i..i ∈ Fin(ℤ)
    using int_ord_is_refl Int_ZF_2_L4 Order_ZF_2_L4 by simp
  moreover from A2 have
    ∀m. i≤m ∧ i..m ∈ Fin(ℤ) ⟶ i..(m $+ $#1) ∈ Fin(ℤ)
    using Int_ZF_4_L4 by simp
  ultimately show i..k ∈ Fin(ℤ) by (rule Int_ZF_3_L7)
next assume ¬ i ≤ k
  then show i..k ∈ Fin(ℤ) using Int_ZF_2_L6 Order_ZF_2_L5
    by simp
qed
```

Bounded integer sets are finite.

```
lemma (in int0) Int_ZF_4_L6: assumes A1: IsBounded(A,IntegerOrder)
  shows A ∈ Fin(ℤ)
proof -
  have T1: ∀m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder).
    $#0..m ∈ Fin(ℤ)
  proof
    fix m assume m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder)
    then have m∈ℤ using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L4E
      by auto
    then show $#0..m ∈ Fin(ℤ) using Int_ZF_4_L5 by simp
  qed
  have group3(ℤ,IntegerAddition,IntegerOrder)
    using Int_ZF_2_T1 by simp
  moreover from T1 have ∀m ∈ Nonnegative(ℤ,IntegerAddition,IntegerOrder).
    Interval(IntegerOrder,TheNeutralElement(ℤ,IntegerAddition),m)
    ∈ Fin(ℤ) using Int_ZF_1_L8 by simp
  moreover from A1 have IsBounded(A,IntegerOrder) .
  ultimately show A ∈ Fin(ℤ) by (rule group3.OrderedGroup_ZF_2_T1)
qed
```

A subset of integers is bounded iff it is finite.

```
theorem (in int0) Int_bounded_iff_fin:
  shows IsBounded(A,IntegerOrder)⟷ A∈Fin(ℤ)
  using Int_ZF_4_L6 Int_ZF_2_T1 group3.ord_group_fin_bounded
  by blast
```

The image of an interval by any integer function is finite, hence bounded.

```
lemma (in int0) Int_ZF_4_L8:
  assumes A1: i∈ℤ  k∈ℤ and A2: f:ℤ→ℤ
  shows
  f(i..k) ∈ Fin(ℤ)
```

```
  IsBounded(f(i..k),IntegerOrder)
  using prems Int_ZF_4_L5 Finite1_L6A Int_bounded_iff_fin
  by auto
```

If for every integer we can find one in A that is greater or equal, then A is is not bounded above, hence infinite.

```
lemma (in int0) Int_ZF_4_L9: assumes A1: ∀m∈ℤ. ∃k∈A. m≤k
  shows
  ¬IsBoundedAbove(A,IntegerOrder)
  A ∉ Fin(ℤ)
proof -
  have ℤ ≠ {0}
    using Int_ZF_1_L8A int_zero_not_one by blast
  with A1 show
    ¬IsBoundedAbove(A,IntegerOrder)
    A ∉ Fin(ℤ)
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L2A
    by auto
qed


end
```

24 Int_ZF_1.thy

```
theory Int_ZF_1 imports Int_ZF OrderedRing_ZF

begin
```

This theory file considers the set of integers as an ordered ring.

24.1 Integers as a ring

In this section we show that integers form a commutative ring.

The next lemma provides the condition to show that addition is distributive with respect to multiplication.

```
lemma (in int0) Int_ZF_1_1_L1: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a·(b+c) = a·b + a·c
  (b+c)·a = b·a + c·a
  using prems Int_ZF_1_L2 zadd_zmult_distrib zadd_zmult_distrib2
  by auto
```

Integers form a commutative ring, hence we can use theorems proven in `ring0` context (locale).

```
lemma (in int0) Int_ZF_1_1_L2: shows
  IsAring(ℤ,IntegerAddition,IntegerMultiplication)
  IntegerMultiplication {is commutative on} ℤ
  ring0(ℤ,IntegerAddition,IntegerMultiplication)
proof -
  have ∀a∈ℤ.∀b∈ℤ.∀c∈ℤ.
    a·(b+c) = a·b + a·c ∧ (b+c)·a = b·a + c·a
    using Int_ZF_1_1_L1 by simp
  then have IsDistributive(ℤ,IntegerAddition,IntegerMultiplication)
    using IsDistributive_def by simp
  then show IsAring(ℤ,IntegerAddition,IntegerMultiplication)
    ring0(ℤ,IntegerAddition,IntegerMultiplication)
    using Int_ZF_1_T1 Int_ZF_1_T2 IsAring_def ring0_def
    by auto
  have ∀a∈ℤ.∀b∈ℤ. a·b = b·a using Int_ZF_1_L4 by simp
  then show IntegerMultiplication {is commutative on} ℤ
    using IsCommutative_def by simp
qed
```

Zero and one are integers.

```
lemma (in int0) int_zero_one_are_int: shows 0∈ℤ  1∈ℤ
  using Int_ZF_1_1_L2 ring0.Ring_ZF_1_L2 by auto
```

Negative of zero is zero.

```
lemma (in int0) int_zero_one_are_intA: shows (-0) = 0
```

```
  using Int_ZF_1_T2 group0.group_inv_of_one by simp
```

Properties with one integer.

```
lemma (in int0) Int_ZF_1_1_L4: assumes A1: a ∈ ℤ
  shows
  a+0 = a
  0+a = a
  a·1 = a   1·a = a
  0·a = 0   a·0 = 0
  (-a) ∈ ℤ  (-(-a)) = a
  a-a = 0   a-0 = a  2·a = a+a
proof -
  from A1 show
    a+0 = a   0+a = a   a·1 = a
    1·a = a   a-a = 0   a-0 = a
    (-a) ∈ ℤ  2·a = a+a   (-(-a)) = a
    using Int_ZF_1_1_L2 ring0.Ring_ZF_1_L3 by auto
  from A1 show 0·a = 0   a·0 = 0
    using Int_ZF_1_1_L2 ring0.Ring_ZF_1_L6 by auto
qed
```

Properties that require two integers.

```
lemma (in int0) Int_ZF_1_1_L5: assumes A1: a∈ℤ  b∈ℤ
  shows
  a+b ∈ ℤ
  a-b ∈ ℤ
  a·b ∈ ℤ
  a+b = b+a
  a·b = b·a
  (-b)-a = (-a)-b
  (-(a+b)) = (-a)-b
  (-(a-b)) = ((-a)+b)
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  (-a)·(-b) = a·b
  using prems Int_ZF_1_1_L2 ring0.Ring_ZF_1_L4 ring0.Ring_ZF_1_L9
    ring0.Ring_ZF_1_L7 ring0.Ring_ZF_1_L7A Int_ZF_1_L4 by auto
```

2 and 3 are integers.

```
lemma (in int0) int_two_three_are_int: shows 2 ∈ ℤ  3 ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 by auto
```

Another property with two integers.

```
lemma (in int0) Int_ZF_1_1_L5B:
  assumes A1: a∈ℤ  b∈ℤ
  shows a-(-b) = a+b
  using prems Int_ZF_1_1_L2 ring0.Ring_ZF_1_L9
  by simp
```

Properties that require three integers.

```
lemma (in int0) Int_ZF_1_1_L6: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a-(b+c) = a-b-c
  a-(b-c) = a-b+c
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using prems Int_ZF_1_1_L2 ring0.Ring_ZF_1_L10  ring0.Ring_ZF_1_L8
  by auto
```

One more property with three integers.

```
lemma (in int0) Int_ZF_1_1_L6A: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows a+(b-c) = a+b-c
  using prems Int_ZF_1_1_L2 ring0.Ring_ZF_1_L10A by simp
```

Associativity of addition and multiplication.

```
lemma (in int0) Int_ZF_1_1_L7: assumes A1: a∈ℤ b∈ℤ c∈ℤ
  shows
  a+b+c = a+(b+c)
  a·b·c = a·(b·c)
  using prems Int_ZF_1_1_L2 ring0.Ring_ZF_1_L11 by auto
```

24.2 Rearrangement lemmas

In this section we collect lemmas about identities related to rearranging the terms in expresssions

A formula with a positive integer.

```
lemma (in int0) Int_ZF_1_2_L1: assumes 0≤a
  shows abs(a)+1 = abs(a+1)
  using prems Int_ZF_2_L16 Int_ZF_2_L12A by simp
```

A formula with two integers, one positive.

```
lemma (in int0) Int_ZF_1_2_L2: assumes A1: a∈ℤ and A2: 0≤b
  shows a+(abs(b)+1)·a = (abs(b+1)+1)·a
proof -
  from A2 have T1: abs(b+1) ∈ ℤ
    using Int_ZF_2_L12A Int_ZF_2_L1A Int_ZF_2_L14 by blast
  with A1 A2 show thesis
    using Int_ZF_1_2_L1 Int_ZF_1_1_L2 ring0.Ring_ZF_2_L1
    by simp
qed
```

A couple of formulae about canceling opposite integers.

```
lemma (in int0) Int_ZF_1_2_L3: assumes A1: a∈ℤ  b∈ℤ
  shows
  a+b-a = b
```

```
  a+(b-a) = b
  a+b-b = a
  a-b+b = a
  (-a)+(a+b) = b
  a+(b-a) = b
  (-b)+(a+b) = a
  a-(b+a) = -b
  a-(a+b) = -b
  a-(a-b) = b
  a-b-a =  -b
  a-b - (a+b) = (-b)-b
  using prems Int_ZF_1_T2 group0.group0_4_L6A group0.group0_2_L16
    group0.group0_2_L16A group0.group0_4_L6AA group0.group0_4_L6AB
    group0.group0_4_L6F group0.group0_4_L6AC by auto
```

Subtracting one does not increase integers. This may be moved to a theory about ordered rings one day.

```
lemma (in int0) Int_ZF_1_2_L3A: assumes A1: a≤b
  shows a-1 ≤ b
proof -
  from A1 have b+1-1 = b
    using Int_ZF_2_L1A int_zero_one_are_int Int_ZF_1_2_L3 by simp
  moreover from A1 have a-1 ≤ b+1-1
    using Int_ZF_2_L12A int_zero_one_are_int Int_ZF_1_1_L4 int_ord_transl_inv
    by simp
  ultimately show a-1 ≤ b by simp
qed
```

Subtracting one does not increase integers, special case.

```
lemma (in int0) Int_ZF_1_2_L3AA:
  assumes A1: a∈ℤ shows
  a-1 ≤a
  a-1 ≠ a
  ¬(a≤a-1)
  ¬(a+1 ≤a)
  ¬(1+a ≤a)
proof -
  from A1 have a≤a using int_ord_is_refl refl_def
    by simp
  then show a-1 ≤a using Int_ZF_1_2_L3A
    by simp
  moreover from A1 show a-1 ≠ a using Int_ZF_1_L14 by simp
  ultimately show I: ¬(a≤a-1) using Int_ZF_2_L19AA
    by blast
  with A1 show ¬(a+1 ≤a)
    using int_zero_one_are_int Int_ZF_2_L9B by simp
  with A1 show ¬(1+a ≤a)
    using int_zero_one_are_int Int_ZF_1_1_L5 by simp
qed
```

A formula with a nonpositive integer.

```
lemma (in int0) Int_ZF_1_2_L4: assumes a≤0
  shows abs(a)+1 = abs(a-1)
  using prems int_zero_one_are_int Int_ZF_1_2_L3A Int_ZF_2_T1
      group3.OrderedGroup_ZF_3_L3A Int_ZF_2_L1A
      int_zero_one_are_int Int_ZF_1_1_L5 by simp
```

A formula with two integers, one negative.

```
lemma (in int0) Int_ZF_1_2_L5: assumes A1: a∈ℤ and A2: b≤0
  shows a+(abs(b)+1)·a = (abs(b-1)+1)·a
proof -
  from A2 have abs(b-1) ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_2_L3A Int_ZF_2_L1A Int_ZF_2_L14

    by blast
  with A1 A2 show thesis
    using Int_ZF_1_2_L4 Int_ZF_1_1_L2 ring0.Ring_ZF_2_L1
    by simp
qed
```

A rearrangement with four integers.

```
lemma (in int0) Int_ZF_1_2_L6:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows
  a-(b-1)·c = (d-b·c)-(d-a-c)
proof -
  from A1 have T1:
    (d-b·c) ∈ ℤ d-a ∈ ℤ (-(b·c)) ∈ ℤ
    using Int_ZF_1_1_L5 Int_ZF_1_1_L4 by auto
  with A1 have
    (d-b·c)-(d-a-c) = (-(b·c))+a+c
    using Int_ZF_1_1_L6 Int_ZF_1_2_L3 by simp
  also from A1 T1 have (-(b·c))+a+c = a-(b-1)·c
    using int_zero_one_are_int Int_ZF_1_1_L6 Int_ZF_1_1_L4 Int_ZF_1_1_L5
    by simp
  finally show thesis by simp
qed
```

Some other rearrangements with two integers.

```
lemma (in int0) Int_ZF_1_2_L7: assumes a∈ℤ  b∈ℤ
  shows
  a·b = (a-1)·b+b
  a·(b+1) = a·b+a
  (b+1)·a = b·a+a
  (b+1)·a = a+b·a
  using prems Int_ZF_1_1_L1 Int_ZF_1_1_L5 int_zero_one_are_int
    Int_ZF_1_1_L6 Int_ZF_1_1_L4 Int_ZF_1_T2 group0.group0_2_L16
  by auto
```

Another rearrangement with two integers.

```
lemma (in int0) Int_ZF_1_2_L8:
  assumes A1: a∈ℤ b∈ℤ
  shows a+1+(b+1) = b+a+2
  using prems int_zero_one_are_int Int_ZF_1_T2 group0.group0_4_L8
  by simp
```

A couple of rearrangement with three integers.

```
lemma (in int0) Int_ZF_1_2_L9:
  assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  (a-b)+(b-c) = a-c
  (a-b)-(a-c) = c-b
  a+(b+(c-a-b)) = c
  (-a)-b+c = c-a-b
  (-b)-a+c = c-a-b
  (-((-a)+b+c)) = a-b-c
  a+b+c-a = b+c
  a+b-(a+c) = b-c
  using prems Int_ZF_1_T2
    group0.group0_4_L4B group0.group0_4_L6D group0.group0_4_L4D
    group0.group0_4_L6B group0.group0_4_L6E
  by auto
```

Another couple of rearrangements with three integers.

```
lemma (in int0) Int_ZF_1_2_L9A:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows (-(a-b-c)) = c+b-a
proof -
  from A1 have T:
    a-b ∈ ℤ  (-(a-b)) ∈ ℤ  (-b) ∈ ℤ using
    Int_ZF_1_1_L4 Int_ZF_1_1_L5 by auto
  with A1 have (-(a-b-c)) = c - ((-b)+a)
     using Int_ZF_1_1_L5 by simp
   also from A1 T have ... = c+b-a
     using Int_ZF_1_1_L6 Int_ZF_1_1_L5B
     by simp
  finally show (-(a-b-c)) = c+b-a
    by simp
qed
```

Another rearrangement with three integers.

```
lemma (in int0) Int_ZF_1_2_L10:
  assumes A1: a∈ℤ b∈ℤ c∈ℤ
  shows (a+1)·b + (c+1)·b = (c+a+2)·b
proof -
  from A1 have a+1 ∈ ℤ c+1 ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 by auto
```

```
  with A1 have
    (a+1)·b + (c+1)·b = (a+1+(c+1))·b
    using Int_ZF_1_1_L1 by simp
  also from A1 have ... = (c+a+2)·b
    using Int_ZF_1_2_L8 by simp
  finally show thesis by simp
qed
```

A technical rearrangement involing inequalities with absolute value.

```
lemma (in int0) Int_ZF_1_2_L10A:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ  e∈ℤ
  and A2: abs(a·b-c) ≤ d  abs(b·a-e) ≤ f
  shows abs(c-e) ≤  f+d
proof -
  from A1 A2 have T1:
    d∈ℤ  f∈ℤ  a·b ∈ ℤ  a·b-c ∈ ℤ  b·a-e ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  with A2 have
    abs((b·a-e)-(a·b-c)) ≤ f +d
    using Int_ZF_2_L21 by simp
  with A1 T1 show abs(c-e) ≤ f+d
    using Int_ZF_1_1_L5 Int_ZF_1_2_L9 by simp
qed
```

Some arithmetics.

```
lemma (in int0) Int_ZF_1_2_L11: assumes A1: a∈ℤ
  shows
  a+1+2 = a+3
  a = 2·a - a
proof -
  from A1 show a+1+2 = a+3
    using int_zero_one_are_int int_two_three_are_int Int_ZF_1_T2 group0.group0_4_L4C
    by simp
  from A1 show a = 2·a - a
    using int_zero_one_are_int Int_ZF_1_1_L1 Int_ZF_1_1_L4 Int_ZF_1_T2
group0.group0_2_L16
    by simp
qed
```

A simple rearrangement with three integers.

```
lemma (in int0) Int_ZF_1_2_L12:
  assumes a∈ℤ  b∈ℤ  c∈ℤ
  shows
  (b-c)·a = a·b - a·c
  using prems Int_ZF_1_1_L6 Int_ZF_1_1_L5 by simp
```

A big rearrangement with five integers.

```
lemma (in int0) Int_ZF_1_2_L13:
```

```
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ  x∈ℤ
  shows (x+(a·x+b)+c)·d = d·(a+1)·x + (b·d+c·d)
proof -
  from A1 have T1:
    a·x ∈ ℤ   (a+1)·x ∈ ℤ
    (a+1)·x + b ∈ ℤ
    using Int_ZF_1_1_L5 int_zero_one_are_int by auto
  with A1 have (x+(a·x+b)+c)·d = ((a+1)·x + b)·d + c·d
    using Int_ZF_1_1_L7 Int_ZF_1_2_L7 Int_ZF_1_1_L1
    by simp
  also from A1 T1 have ... = (a+1)·x·d + b · d + c·d
    using Int_ZF_1_1_L1 by simp
  finally have (x+(a·x+b)+c)·d = (a+1)·x·d + b·d + c·d
    by simp
  moreover from A1 T1 have (a+1)·x·d = d·(a+1)·x
    using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_1_1_L7 by simp
  ultimately have (x+(a·x+b)+c)·d = d·(a+1)·x + b·d + c·d
    by simp
  moreover from A1 T1 have
    d·(a+1)·x ∈ ℤ  b·d ∈ ℤ  c·d ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 by auto
  ultimately show thesis using Int_ZF_1_1_L7 by simp
qed
```

Rerrangement about adding linear functions.

```
lemma (in int0) Int_ZF_1_2_L14:
  assumes a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ  x∈ℤ
  shows (a·x + b) + (c·x + d) = (a+c)·x + (b+d)
  using prems Int_ZF_1_1_L2 ring0.Ring_ZF_2_L3 by simp
```

A rearrangement with four integers. Again we have to use the generic set notation to use a theorem proven in different context.

```
lemma (in int0) Int_ZF_1_2_L15: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ
  and A2: a = b-c-d
  shows
  d = b-a-c
  d = (-a)+b-c
  b = a+d+c
proof -
  let G = int
  let f = IntegerAddition
  from A1 A2 have I:
    group0(G, f)   f {is commutative on} G
    a ∈ G  b ∈ G c ∈ G  d ∈ G
    a = f⟨f⟨b,GroupInv(G, f)(c)⟩,GroupInv(G, f)(d)⟩
    using Int_ZF_1_T2 by auto
  then have
    d = f⟨f⟨b,GroupInv(G, f)(a)⟩,GroupInv(G,f)(c)⟩
    by (rule group0.group0_4_L9)
```

```
  then show d = b-a-c by simp
  from I have d = f⟨f⟨GroupInv(G, f)(a),b⟩, GroupInv(G, f)(c)⟩
    by (rule group0.group0_4_L9)
  thus d = (-a)+b-c
    by simp
  from I have b = f⟨f⟨a, d⟩,c⟩
    by (rule group0.group0_4_L9)
  thus b = a+d+c by simp
qed
```

A rearrangement with four integers. Property of groups.

```
lemma (in int0) Int_ZF_1_2_L16:
  assumes a∈ℤ  b∈ℤ  c∈ℤ d∈ℤ
  shows a+(b-c)+d = a+b+d-c
  using prems Int_ZF_1_T2 group0.group0_4_L8 by simp
```

Some rearrangements with three integers. Properties of groups.

```
lemma (in int0) Int_ZF_1_2_L17:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  a+b-c+(c-b) = a
  a+(b+c)-c = a+b
proof -
  let G = int
  let f = IntegerAddition
  from A1 have I:
    group0(G, f)
    a ∈ G  b ∈ G c ∈ G
    using Int_ZF_1_T2 by auto
  then have
    f⟨f⟨f⟨a,b⟩,GroupInv(G, f)(c)⟩,f⟨c,GroupInv(G, f)(b)⟩⟩ = a
    by (rule group0.group0_2_L14A)
  thus a+b-c+(c-b) = a by simp
  from I have
    f⟨f⟨a,f⟨b,c⟩⟩,GroupInv(G, f)(c)⟩ = f⟨a,b⟩
    by (rule group0.group0_2_L14A)
  thus a+(b+c)-c = a+b by simp
qed
```

Another rearrangement with three integers. Property of abelian groups.

```
lemma (in int0) Int_ZF_1_2_L18:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows a+b-c+(c-a) = b
proof -
  let G = int
  let f = IntegerAddition
  from A1 have
    group0(G, f)   f {is commutative on} G
    a ∈ G  b ∈ G c ∈ G
```

```
    using Int_ZF_1_T2 by auto
  then have
    f⟨f⟨f⟨a,b⟩,GroupInv(G, f)(c)⟩,f⟨c,GroupInv(G, f)(a)⟩⟩ = b
    by (rule group0.group0_4_L6D)
  thus a+b-c+(c-a) = b by simp
qed
```

24.3 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. Since we don't have the theory of oredered rings we temporarily put some facts about integers as an ordered ring in this section.

We start with the property that a product of nonnegative integers is nonnegative. The proof is by induction and the next lemma is the induction step.

```
lemma (in int0) Int_ZF_1_3_L1: assumes A1: 0≤a  0≤b
  and A3: 0 ≤ a·b
  shows 0 ≤ a·(b+1)
proof -
  from A1 A3 have 0+0 ≤ a·b+a
    using int_ineq_add_sides by simp
  with A1 show 0 ≤ a·(b+1)
    using int_zero_one_are_int Int_ZF_1_1_L4 Int_ZF_2_L1A Int_ZF_1_2_L7

    by simp
qed
```

Product of nonnegative integers is nonnegative.

```
lemma (in int0) Int_ZF_1_3_L2: assumes A1: 0≤a  0≤b
  shows 0≤a·b
proof -
  from A1 have 0≤b by simp
  moreover from A1 have 0 ≤ a·0 using
    Int_ZF_2_L1A Int_ZF_1_1_L4 int_zero_one_are_int int_ord_is_refl refl_def
    by simp
  moreover from A1 have
    ∀m. 0≤m ∧ 0≤a·m ⟶ 0 ≤ a·(m+1)
    using Int_ZF_1_3_L1 by simp
  ultimately show 0≤a·b by (rule Induction_on_int)
qed
```

The set of nonnegative integers is closed under multiplication.

```
lemma (in int0) Int_ZF_1_3_L2A: shows
  ℤ⁺ {is closed under} IntegerMultiplication
proof -
```

```
  { fix a b assume a∈ℤ⁺  b∈ℤ⁺
    then have a·b ∈ℤ⁺
      using Int_ZF_1_3_L2 Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L2
      by simp
  } then have ∀a∈ℤ⁺.∀b∈ℤ⁺.a·b ∈ℤ⁺ by simp
  then show thesis using IsOpClosed_def by simp
qed
```

Integers form an ordered ring. All theorems proven in the `ring1` context are valid in `int0` context.

```
theorem (in int0) Int_ZF_1_3_T1: shows
  IsAnOrdRing(ℤ,IntegerAddition,IntegerMultiplication,IntegerOrder)
  ring1(ℤ,IntegerAddition,IntegerMultiplication,IntegerOrder)
  using Int_ZF_1_1_L2 Int_ZF_2_L1B Int_ZF_1_3_L2A Int_ZF_2_T1
    OrdRing_ZF_1_L6 OrdRing_ZF_1_L2 by auto
```

Product of integers that are greater that one is greater than one. The proof is by induction and the next step is the induction step.

```
lemma (in int0) Int_ZF_1_3_L3_indstep:
  assumes A1: 1≤a  1≤b
  and A2: 1 ≤ a·b
  shows 1 ≤ a·(b+1)
proof -
   from A1 A2 have 1≤2 and 2 ≤ a·(b+1)
    using Int_ZF_2_L1A int_ineq_add_sides Int_ZF_2_L16B Int_ZF_1_2_L7

    by auto
  then show 1 ≤ a·(b+1) by (rule Int_order_transitive)
qed
```

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

```
lemma (in int0) Int_ZF_1_3_L3:
  assumes A1: 1≤a 1≤b
  shows 1 ≤ a·b
proof -
  from A1 have 1≤b  1≤a·1
    using Int_ZF_2_L1A Int_ZF_1_1_L4 by auto
  moreover from A1 have
    ∀m. 1≤m ∧ 1 ≤ a·m ⟶ 1 ≤ a·(m+1)
    using Int_ZF_1_3_L3_indstep by simp
  ultimately show 1 ≤ a·b by (rule Induction_on_int)
qed
```

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

```
lemma (in int0) Int_ZF_1_3_L4: assumes a∈ℤ  b∈ℤ
  shows
  abs((-a)·b) = abs(a·b)
```

```
  abs(a·(-b)) = abs(a·b)
  abs((-a)·(-b)) = abs(a·b)
  using prems Int_ZF_1_1_L5 Int_ZF_2_L17 by auto
```

Absolute value of a product is the product of absolute values. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L5:
  assumes A1: a∈ℤ  b∈ℤ
  shows abs(a·b) = abs(a)·abs(b)
  using prems Int_ZF_1_3_T1 ring1.OrdRing_ZF_2_L5 by simp
```

Double nonnegative is nonnegative. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L5A: assumes 0≤a
  shows 0≤2·a
  using prems Int_ZF_1_3_T1 ring1.OrdRing_ZF_1_L5A by simp
```

The next lemma shows what happens when one integer is not greater or equal than another.

```
lemma (in int0) Int_ZF_1_3_L6:
  assumes A1: a∈ℤ  b∈ℤ
  shows ¬(b≤a) ⟷ a+1 ≤ b
proof
  assume A3: ¬(b≤a)
  with A1 have  a≤b by (rule Int_ZF_2_L19)
  then have a = b   ∨  a+1 ≤ b
    using Int_ZF_4_L1B by simp
  moreover from A1 A3 have a≠b by (rule Int_ZF_2_L19)
  ultimately show a+1 ≤ b by simp
next assume A4: a+1 ≤ b
  { assume b≤a
    with A4 have a+1 ≤ a by (rule Int_order_transitive)
    moreover from A1 have a ≤ a+1
      using Int_ZF_2_L12B by simp
    ultimately have a+1 = a
      by (rule Int_ZF_2_L3)
    with A1 have False using Int_ZF_1_L14 by simp
  } then show ¬(b≤a) by auto
qed
```

Another form of stating that there are no integers between integers m and $m + 1$.

```
corollary (in int0) no_int_between: assumes A1: a∈ℤ  b∈ℤ
  shows b≤a ∨ a+1 ≤ b
  using A1 Int_ZF_1_3_L6 by auto
```

Another way of saying what it means that one integer is not greater or equal than another.

```
corollary (in int0) Int_ZF_1_3_L6A:
```

```
  assumes A1: a∈ℤ  b∈ℤ and A2: ¬(b≤a)
  shows a ≤ b-1
proof -
  from A1 A2 have a+1 - 1 ≤ b - 1
    using Int_ZF_1_3_L6 int_zero_one_are_int Int_ZF_1_1_L4
      int_ord_transl_inv by simp
  with A1 show a ≤ b-1
    using int_zero_one_are_int Int_ZF_1_2_L3
    by simp
qed
```

Yet another form of stating that there are nointegers between m and $m+1$.

```
lemma (in int0) no_int_between1:
  assumes A1: a≤b  and A2: a≠b
  shows
  a+1 ≤ b
  a ≤ b-1
proof -
  from A1 have T: a∈ℤ  b∈ℤ using Int_ZF_2_L1A
    by auto
  { assume b≤a
    with A1 have a=b by (rule Int_ZF_2_L3)
    with A2 have False by simp }
  then have ¬(b≤a) by auto
  with T show
    a+1 ≤ b
    a ≤ b-1
    using no_int_between Int_ZF_1_3_L6A by auto
qed
```

We can decompose proofs into three cases: $a = b$, $a \leq b - 1b$ or $a \geq b + 1b$.

```
lemma (in int0) Int_ZF_1_3_L6B: assumes A1: a∈ℤ  b∈ℤ
  shows a=b ∨ (a ≤ b-1) ∨ (b+1 ≤a)
proof -
  from A1 have a=b ∨ (a≤b ∧ a≠b) ∨ (b≤a ∧ b≠a)
    using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L31
    by simp
  then show thesis using no_int_between1
    by auto
qed
```

A special case of `Int_ZF_1_3_L6B` when $b = 0$. This allows to split the proofs in cases $a \leq -1$, $a = 0$ and $a \geq 1$.

```
corollary (in int0) Int_ZF_1_3_L6C: assumes A1: a∈ℤ
  shows a=0 ∨ (a ≤ -1) ∨ (1≤a)
proof -
  from A1 have a=0 ∨ (a ≤ 0 -1) ∨ (0 +1 ≤a)
    using int_zero_one_are_int Int_ZF_1_3_L6B by simp
  then show thesis using Int_ZF_1_1_L4 int_zero_one_are_int
```

```
    by simp
qed
```

An integer is not less or equal zero iff it is greater or equal one.

```
lemma (in int0) Int_ZF_1_3_L7: assumes a∈ℤ
  shows ¬(a≤0) ⟷ 1 ≤ a
  using prems int_zero_one_are_int Int_ZF_1_3_L6 Int_ZF_1_1_L4
  by simp
```

Product of positive integers is positive.

```
lemma (in int0) Int_ZF_1_3_L8:
  assumes a∈ℤ  b∈ℤ
  and ¬(a≤0)  ¬(b≤0)
  shows ¬((a·b) ≤ 0)
  using prems Int_ZF_1_3_L7 Int_ZF_1_3_L3 Int_ZF_1_1_L5 Int_ZF_1_3_L7
  by simp
```

If $a \cdot b$ is nonnegative and b is positive, then a is nonnegative. Proof by contradiction.

```
lemma (in int0) Int_ZF_1_3_L9:
  assumes A1: a∈ℤ  b∈ℤ
  and A2:  ¬(b≤0) and A3: a·b ≤ 0
  shows a≤0
proof -
  { assume ¬(a≤0)
    with A1 A2 have ¬((a·b) ≤ 0) using Int_ZF_1_3_L8
      by simp
  } with A3 show a≤0 by auto
qed
```

One integer is less or equal another iff the difference is nonpositive.

```
lemma (in int0) Int_ZF_1_3_L10:
  assumes a∈ℤ  b∈ℤ
  shows a≤b ⟷ a-b ≤ 0
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L9
  by simp
```

Some conclusions from the fact that one integer is less or equal than another.

```
lemma (in int0) Int_ZF_1_3_L10A: assumes a≤b
  shows 0 ≤ b-a
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L12A
  by simp
```

We can simplify out a positive element on both sides of an inequality.

```
lemma (in int0) Int_ineq_simpl_positive:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  and A2: a·c ≤ b·c and A4: ¬(c≤0)
  shows a ≤ b
```

```
proof -
  from A1 A4 have a-b ∈ ℤ  c∈ℤ  ¬(c≤0)
    using Int_ZF_1_1_L5 by auto
  moreover from A1 A2 have (a-b)·c ≤ 0
    using Int_ZF_1_1_L5 Int_ZF_1_3_L10 Int_ZF_1_1_L6
    by simp
  ultimately have a-b ≤ 0 by (rule Int_ZF_1_3_L9)
  with A1 show a ≤ b using Int_ZF_1_3_L10 by simp
qed
```

A technical lemma about conclusion from an inequality between absolute values. This is a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L11:
  assumes A1: a∈ℤ  b∈ℤ
  and A2: ¬(abs(a) ≤ abs(b))
  shows ¬(abs(a) ≤ 0)
proof -
  { assume abs(a) ≤ 0
    moreover from A1 have 0 ≤ abs(a) using int_abs_nonneg
      by simp
    ultimately have abs(a) = 0 by (rule Int_ZF_2_L3)
    with A1 A2 have False using int_abs_nonneg by simp
  } then show  ¬(abs(a) ≤ 0) by auto
qed
```

Negative times positive is negative. This a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L12:
  assumes a≤0  and 0≤b
  shows a·b ≤ 0
  using prems Int_ZF_1_3_T1 ring1.OrdRing_ZF_1_L8
  by simp
```

We can multiply an inequality by a nonnegative number. This is a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L13:
  assumes A1: a≤b and A2: 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
  using prems Int_ZF_1_3_T1 ring1.OrdRing_ZF_1_L9 by auto
```

A technical lemma about decreasing a factor in an inequality.

```
lemma (in int0) Int_ZF_1_3_L13A:
  assumes 1≤a and b≤c and (a+1)·c ≤ d
  shows (a+1)·b ≤ d
proof -
  from prems have
    (a+1)·b ≤ (a+1)·c
```

```
    (a+1)·c ≤ d
    using Int_ZF_2_L16C Int_ZF_1_3_L13 by auto
  then show (a+1)·b ≤ d by (rule Int_order_transitive)
qed
```

We can multiply an inequality by a positive number. This is a property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L13B:
  assumes A1: a≤b and A2: c∈ℤ+
  shows
  a·c ≤ b·c
  c·a ≤ c·b
proof -
  let R = ℤ
  let A = IntegerAddition
  let M = IntegerMultiplication
  let r = IntegerOrder
  from A1 A2 have
    ring1(R, A, M, r)
    ⟨a,b⟩ ∈ r
    c ∈ PositiveSet(R, A, r)
    using Int_ZF_1_3_T1 by auto
  then show
    a·c ≤ b·c
    c·a ≤ c·b
    using ring1.OrdRing_ZF_1_L9A by auto
qed
```

A rearrangement with four integers and absolute value.

```
lemma (in int0) Int_ZF_1_3_L14:
  assumes  A1: a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows abs(a·b)+(abs(a)+c)·d = (d+abs(b))·abs(a)+c·d
proof -
  from A1 have T1:
    abs(a) ∈ ℤ  abs(b) ∈ ℤ
    abs(a)·abs(b) ∈ ℤ
    abs(a)·d ∈ ℤ
    c·d ∈ ℤ
    abs(b)+d ∈ ℤ
    using Int_ZF_2_L14 Int_ZF_1_1_L5 by auto
  with A1 have abs(a·b)+(abs(a)+c)·d = abs(a)·(abs(b)+d)+c·d
    using Int_ZF_1_3_L5 Int_ZF_1_1_L1 Int_ZF_1_1_L7 by simp
  with A1 T1 show thesis using Int_ZF_1_1_L5 by simp
qed
```

A technical lemma about what happens when one absolute value is not greater or equal than another.

```
lemma (in int0) Int_ZF_1_3_L15: assumes A1: m∈ℤ n∈ℤ
```

```
  and A2: ¬(abs(m) ≤ abs(n))
  shows n ≤ abs(m)  m≠0
proof -
  from A1 have T1: n ≤ abs(n)
    using Int_ZF_2_L19C by simp
  from A1 have abs(n) ∈ ℤ  abs(m) ∈ ℤ
    using Int_ZF_2_L14 by auto
  moreover from A2 have ¬(abs(m) ≤ abs(n)) .
  ultimately have abs(n) ≤ abs(m)
    by (rule Int_ZF_2_L19)
  with T1 show  n ≤ abs(m) by (rule Int_order_transitive)
  from A1 A2 show m≠0 using Int_ZF_2_L18 int_abs_nonneg by auto
qed
```

Negative of a nonnegative is nonpositive.

```
lemma (in int0) Int_ZF_1_3_L16: assumes A1: 0 ≤ m
  shows (-m) ≤ 0
proof -
  from A1 have (-m) ≤ (-0)
    using Int_ZF_2_L10 by simp
  then show (-m) ≤ 0 using Int_ZF_1_L11
    by simp
qed
```

Some statements about intervals centered at 0.

```
lemma (in int0) Int_ZF_1_3_L17: assumes A1: m∈ℤ
  shows
  (-abs(m)) ≤ abs(m)
  (-abs(m))..abs(m) ≠ 0
proof -
  from A1 have (-abs(m)) ≤ 0  0 ≤ abs(m)
    using int_abs_nonneg Int_ZF_1_3_L16 by auto
  then show (-abs(m)) ≤ abs(m) by (rule Int_order_transitive)
  then have abs(m) ∈ (-abs(m))..abs(m)
    using int_ord_is_refl Int_ZF_2_L1A Order_ZF_2_L2 by simp
  thus (-abs(m))..abs(m) ≠ 0 by auto
qed
```

The greater of two integers is indeed greater than both, and the smaller one is smaller that both.

```
lemma (in int0) Int_ZF_1_3_L18: assumes A1: m∈ℤ  n∈ℤ
  shows
  m ≤ GreaterOf(IntegerOrder,m,n)
  n ≤ GreaterOf(IntegerOrder,m,n)
  SmallerOf(IntegerOrder,m,n) ≤ m
  SmallerOf(IntegerOrder,m,n) ≤ n
  using prems Int_ZF_2_T1 Order_ZF_3_L2 by auto
```

If $|m| \leq n$, then $m \in -n..n$.

```
lemma (in int0) Int_ZF_1_3_L19:
  assumes A1: m∈ℤ and A2: abs(m) ≤ n
  shows
  (-n) ≤ m  m ≤ n
  m ∈ (-n)..n
  0 ≤ n
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L8
    group3.OrderedGroup_ZF_3_L8A Order_ZF_2_L1
  by auto
```

A slight generalization of the above lemma.

```
lemma (in int0) Int_ZF_1_3_L19A:
  assumes A1: m∈ℤ and A2: abs(m) ≤ n and A3: 0≤k
  shows (-(n+k)) ≤ m
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L8B
  by simp
```

Sets of integers that have absolute value bounded are bounded.

```
lemma (in int0) Int_ZF_1_3_L20:
  assumes A1: ∀x∈X. b(x) ∈ ℤ ∧ abs(b(x)) ≤ L
  shows IsBounded({b(x). x∈X},IntegerOrder)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 have
    group3(G, P, r)
    r {is total on} G
    ∀x∈X. b(x) ∈ G ∧ ⟨AbsoluteValue(G, P, r)  b(x), L⟩ ∈ r
    using Int_ZF_2_T1 by auto
  then show IsBounded({b(x). x∈X},IntegerOrder)
    by (rule group3.OrderedGroup_ZF_3_L9A)
qed
```

If a set is bounded, then the absolute values of the elements of that set are bounded.

```
lemma (in int0) Int_ZF_1_3_L20A: assumes IsBounded(A,IntegerOrder)
  shows ∃L. ∀a∈A. abs(a) ≤ L
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L10A
  by simp
```

Absolute vaues of integers from a finite image of integers are bounded by an integer.

```
lemma (in int0) Int_ZF_1_3_L20AA:
  assumes A1: {b(x). x∈ℤ} ∈ Fin(ℤ)
  shows ∃L∈ℤ. ∀x∈ℤ. abs(b(x)) ≤ L
  using prems int_not_empty Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L11A
  by simp
```

If absolute values of values of some integer function are bounded, then the image a set from the domain is a bounded set.

```
lemma (in int0) Int_ZF_1_3_L20B:
  assumes f:X→ℤ and A⊆X and ∀x∈A. abs(f(x)) ≤ L
  shows  IsBounded(f(A),IntegerOrder)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from prems have
    group3(G, P, r)
    r {is total on} G
    f:X→G
    A⊆X
    ∀x∈A. ⟨AbsoluteValue(G, P, r)(f(x)), L⟩ ∈ r
    using Int_ZF_2_T1 by auto
  then show IsBounded(f(A), r)
    by (rule group3.OrderedGroup_ZF_3_L9B)
qed
```

A special case of the previous lemma for a function from integers to integers.

```
corollary (in int0) Int_ZF_1_3_L20C:
  assumes f:ℤ→ℤ and ∀m∈ℤ. abs(f(m)) ≤ L
  shows f(ℤ) ∈ Fin(ℤ)
proof -
  from prems have f:ℤ→ℤ ℤ ⊆ ℤ  ∀m∈ℤ. abs(f(m)) ≤ L
    by auto
  then have IsBounded(f(ℤ),IntegerOrder)
    by (rule Int_ZF_1_3_L20B)
  then show f(ℤ) ∈ Fin(ℤ) using Int_bounded_iff_fin
    by simp
qed
```

A triangle inequality with three integers. Property of linearly ordered abelian groups.

```
lemma (in int0) int_triangle_ineq3:
  assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows abs(a-b-c) ≤ abs(a) + abs(b) + abs(c)
proof -
  from A1 have T: a-b ∈ ℤ  abs(c) ∈ ℤ
    using Int_ZF_1_1_L5 Int_ZF_2_L14 by auto
  with A1 have abs(a-b-c) ≤ abs(a-b) + abs(c)
    using Int_triangle_ineq1 by simp
  moreover from A1 T have
    abs(a-b) + abs(c) ≤  abs(a) + abs(b) + abs(c)
    using Int_triangle_ineq1 int_ord_transl_inv by simp
  ultimately show thesis by (rule Int_order_transitive)
qed
```

If $a \leq c$ and $b \leq c$, then $a + b \leq 2 \cdot c$. Property of ordered rings.

```
lemma (in int0) Int_ZF_1_3_L21:
  assumes A1: a≤c  b≤c shows a+b ≤ 2·c
  using prems Int_ZF_1_3_T1 ring1.OrdRing_ZF_2_L6 by simp
```

If an integer a is between b and $b + c$, then $|b - a| \leq c$. Property of ordered groups.

```
lemma (in int0) Int_ZF_1_3_L22:
  assumes a≤b and c∈ℤ and b≤ c+a
  shows abs(b-a) ≤ c
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L8C
  by simp
```

An application of the triangle inequality with four integers. Property of linearly ordered abelian groups.

```
lemma (in int0) Int_ZF_1_3_L22A:
  assumes  a∈ℤ  b∈ℤ  c∈ℤ  d∈ℤ
  shows abs(a-c) ≤ abs(a+b) + abs(c+d) + abs(b-d)
  using prems Int_ZF_1_T2 Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L7F
  by simp
```

If an integer a is between b and $b + c$, then $|b - a| \leq c$. Property of ordered groups. A version of Int_ZF_1_3_L22 with sligtly different assumptions.

```
lemma (in int0) Int_ZF_1_3_L23:
  assumes A1: a≤b and A2: c∈ℤ and A3: b≤ a+c
  shows abs(b-a) ≤ c
proof -
  from A1 have a ∈ ℤ
    using Int_ZF_2_L1A by simp
  with A2 A3 have b≤ c+a
    using Int_ZF_1_1_L5 by simp
  with A1 A2 show abs(b-a) ≤ c
    using Int_ZF_1_3_L22 by simp
qed
```

24.4 Maximum and minimum of a set of integers

In this section we provide some sufficient conditions for integer subsets to have extrema (maxima and minima).

Finite nonempty subsets of integers attain maxima and minima.

```
theorem (in int0) Int_fin_have_max_min:
  assumes A1: A ∈ Fin(ℤ) and A2: A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  HasAminimum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
```

```
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  Maximum(IntegerOrder,A) ∈ ℤ
  Minimum(IntegerOrder,A) ∈ ℤ
proof -
  from A1 have
    A=0 ∨ HasAmaximum(IntegerOrder,A) and
    A=0 ∨ HasAminimum(IntegerOrder,A)
    using Int_ZF_2_T1 Int_ZF_2_L6 Finite_ZF_1_1_T1A Finite_ZF_1_1_T1B
    by auto
  with A2 show
    HasAmaximum(IntegerOrder,A)
    HasAminimum(IntegerOrder,A)
    by auto
  from A1 A2 show
    Maximum(IntegerOrder,A) ∈ A
    Minimum(IntegerOrder,A) ∈ A
    ∀x∈A. x ≤ Maximum(IntegerOrder,A)
    ∀x∈A. Minimum(IntegerOrder,A) ≤ x
    using Int_ZF_2_T1 Finite_ZF_1_T2 by auto
  moreover from A1 have A⊆ℤ using FinD by simp
  ultimately show
    Maximum(IntegerOrder,A) ∈ ℤ
    Minimum(IntegerOrder,A) ∈ ℤ
    by auto
qed
```

Bounded nonempty integer subsets attain maximum and minimum.

```
theorem (in int0) Int_bounded_have_max_min:
  assumes IsBounded(A,IntegerOrder) and A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  HasAminimum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  Maximum(IntegerOrder,A) ∈ ℤ
  Minimum(IntegerOrder,A) ∈ ℤ
  using prems Int_fin_have_max_min Int_bounded_iff_fin
  by auto
```

Nonempty set of integers that is bounded below attains its minimum.

```
theorem (in int0) int_bounded_below_has_min:
  assumes A1: IsBoundedBelow(A,IntegerOrder) and A2: A≠0
  shows
  HasAminimum(IntegerOrder,A)
  Minimum(IntegerOrder,A) ∈ A
```

```
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
proof -
  from A1 A2 have
    IntegerOrder {is total on} ℤ
    trans(IntegerOrder)
    IntegerOrder ⊆ ℤ×ℤ
    ∀A. IsBounded(A,IntegerOrder) ∧ A≠0 ⟶ HasAminimum(IntegerOrder,A)
    A≠0  IsBoundedBelow(A,IntegerOrder)
    using Int_ZF_2_T1 Int_ZF_2_L6 Int_ZF_2_L1B Int_bounded_have_max_min
    by auto
  then show HasAminimum(IntegerOrder,A)
    by (rule Order_ZF_4_L11)
  then show
    Minimum(IntegerOrder,A) ∈ A
    ∀x∈A. Minimum(IntegerOrder,A) ≤ x
    using Int_ZF_2_L4 Order_ZF_4_L4 by auto
qed
```

Nonempty set of integers that is bounded above attains its maximum.

```
theorem (in int0) int_bounded_above_has_max:
  assumes A1: IsBoundedAbove(A,IntegerOrder) and A2: A≠0
  shows
  HasAmaximum(IntegerOrder,A)
  Maximum(IntegerOrder,A) ∈ A
  Maximum(IntegerOrder,A) ∈ ℤ
  ∀x∈A. x ≤ Maximum(IntegerOrder,A)
proof -
  from A1 A2 have
    IntegerOrder {is total on} ℤ
    trans(IntegerOrder) and
    I: IntegerOrder ⊆ ℤ×ℤ and
    ∀A. IsBounded(A,IntegerOrder) ∧ A≠0 ⟶ HasAmaximum(IntegerOrder,A)
    A≠0  IsBoundedAbove(A,IntegerOrder)
    using Int_ZF_2_T1 Int_ZF_2_L6 Int_ZF_2_L1B Int_bounded_have_max_min
    by auto
  then show HasAmaximum(IntegerOrder,A)
    by (rule Order_ZF_4_L11A)
  then show
    II: Maximum(IntegerOrder,A) ∈ A and
    ∀x∈A. x ≤ Maximum(IntegerOrder,A)
    using Int_ZF_2_L4 Order_ZF_4_L3 by auto
  from I A1 have A ⊆ ℤ by (rule Order_ZF_3_L1A)
  with II show Maximum(IntegerOrder,A) ∈ ℤ by auto
qed
```

A set defined by separation over a bounded set attains its maximum and minimum.

```
lemma (in int0) Int_ZF_1_4_L1:
```

```
  assumes A1: IsBounded(A,IntegerOrder) and A2: A≠0
  and A3: ∀q∈ℤ. F(q) ∈ ℤ
  and A4: K = {F(q). q ∈ A}
  shows
  HasAmaximum(IntegerOrder,K)
  HasAminimum(IntegerOrder,K)
  Maximum(IntegerOrder,K) ∈ K
  Minimum(IntegerOrder,K) ∈ K
  Maximum(IntegerOrder,K) ∈ ℤ
  Minimum(IntegerOrder,K) ∈ ℤ
  ∀q∈A. F(q) ≤ Maximum(IntegerOrder,K)
  ∀q∈A. Minimum(IntegerOrder,K) ≤ F(q)
  IsBounded(K,IntegerOrder)
proof -
  from A1 have A ∈ Fin(ℤ) using Int_bounded_iff_fin
    by simp
  with A3 have {F(q). q ∈ A} ∈ Fin(ℤ)
    by (rule Finite1_L6)
  with A2 A4 have T1: K ∈ Fin(ℤ)  K≠0 by auto
  then show T2:
    HasAmaximum(IntegerOrder,K)
    HasAminimum(IntegerOrder,K)
    and Maximum(IntegerOrder,K) ∈ K
    Minimum(IntegerOrder,K) ∈ K
    Maximum(IntegerOrder,K) ∈ ℤ
    Minimum(IntegerOrder,K) ∈ ℤ
    using Int_fin_have_max_min by auto
  { fix q assume q∈A
    with A4 have F(q) ∈ K by auto
    with T1 have
      F(q) ≤ Maximum(IntegerOrder,K)
      Minimum(IntegerOrder,K) ≤ F(q)
      using Int_fin_have_max_min by auto
  } then show
      ∀q∈A. F(q) ≤ Maximum(IntegerOrder,K)
      ∀q∈A. Minimum(IntegerOrder,K) ≤ F(q)
    by auto
  from T2 show IsBounded(K,IntegerOrder)
    using Order_ZF_4_L7 Order_ZF_4_L8A IsBounded_def
    by simp
qed
```

A three element set has a maximume and minimum.

```
lemma (in int0) Int_ZF_1_4_L1A: assumes A1: a∈ℤ  b∈ℤ  c∈ℤ
  shows
  Maximum(IntegerOrder,{a,b,c}) ∈  ℤ
  a ≤ Maximum(IntegerOrder,{a,b,c})
  b ≤ Maximum(IntegerOrder,{a,b,c})
  c ≤ Maximum(IntegerOrder,{a,b,c})
```

```
  using prems Int_ZF_2_T1 Finite_ZF_1_L2A by auto
```

Integer functions attain maxima and minima over intervals.

```
lemma (in int0) Int_ZF_1_4_L2:
  assumes A1: f:ℤ→ℤ and A2: a≤b
  shows
  maxf(f,a..b) ∈ ℤ
  ∀c ∈ a..b. f(c) ≤ maxf(f,a..b)
  ∃c ∈ a..b. f(c) = maxf(f,a..b)
  minf(f,a..b) ∈ ℤ
  ∀c ∈ a..b. minf(f,a..b) ≤ f(c)
  ∃c ∈ a..b. f(c) = minf(f,a..b)
proof -
  from A2 have T: a∈ℤ  b∈ℤ  a..b ⊆ ℤ
    using Int_ZF_2_L1A Int_ZF_2_L1B Order_ZF_2_L6
    by auto
  with A1 A2 have
    Maximum(IntegerOrder,f(a..b)) ∈ f(a..b)
    ∀x∈f(a..b). x ≤ Maximum(IntegerOrder,f(a..b))
    Maximum(IntegerOrder,f(a..b)) ∈ ℤ
    Minimum(IntegerOrder,f(a..b)) ∈ f(a..b)
    ∀x∈f(a..b). Minimum(IntegerOrder,f(a..b)) ≤ x
    Minimum(IntegerOrder,f(a..b)) ∈ ℤ
    using Int_ZF_4_L8 Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L6
      Int_fin_have_max_min by auto
  with A1 T show
    maxf(f,a..b) ∈ ℤ
    ∀c ∈ a..b. f(c) ≤ maxf(f,a..b)
    ∃c ∈ a..b. f(c) = maxf(f,a..b)
    minf(f,a..b) ∈ ℤ
    ∀c ∈ a..b. minf(f,a..b) ≤ f(c)
    ∃c ∈ a..b. f(c) = minf(f,a..b)
    using func_imagedef by auto
qed
```

24.5 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 rephrasse some lemmas about the set of positive integers known from the theory of oredered grups.

The set of positive integers is closed under addition.

```
lemma (in int0) pos_int_closed_add:
  shows ℤ+ {is closed under} IntegerAddition
  using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L13 by simp
```

Text expended version of the fact that the set of positive integers is closed under addition

```
lemma (in int0) pos_int_closed_add_unfolded:
  assumes a∈ℤ₊  b∈ℤ₊  shows a+b ∈ ℤ₊
  using prems pos_int_closed_add IsOpClosed_def
  by simp
```

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

```
lemma (in int0) Int_ZF_1_5_L1: shows
  IsBoundedBelow(ℤ⁺,IntegerOrder)
  IsBoundedBelow(ℤ₊,IntegerOrder)
  using Nonnegative_def PositiveSet_def IsBoundedBelow_def by auto
```

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

```
lemma (in int0) Int_ZF_1_5_L1A: assumes A1: A ⊆ ℤ⁺
  shows IsBoundedBelow(A,IntegerOrder)
  using A1 Int_ZF_1_5_L1 Order_ZF_3_L12 by blast
```

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

```
lemma (in int0) Int_ZF_1_5_L1B: assumes A1: A ⊆ ℤ₊
  shows IsBoundedBelow(A,IntegerOrder)
  using A1 Int_ZF_1_5_L1 Order_ZF_3_L12 by blast
```

Every nonempty subset of positive integers has a mimimum.

```
lemma (in int0) Int_ZF_1_5_L1C: assumes A ⊆ ℤ₊ and A ≠ 0
  shows
  HasAminimum(IntegerOrder,A)
  Minimum(IntegerOrder,A) ∈ A
  ∀x∈A. Minimum(IntegerOrder,A) ≤ x
  using prems Int_ZF_1_5_L1B int_bounded_below_has_min by auto
```

Infinite subsets of Z^+ do not have a maximum - If $A \subseteq Z^+$ then for every integer we can find one in the set that is not smaller.

```
lemma (in int0) Int_ZF_1_5_L2:
  assumes A1: A ⊆ ℤ⁺  and A2: A ∉ Fin(ℤ) and A3: D∈ℤ
  shows ∃n∈A. D≤n
proof -
  { assume ∀n∈A. ¬(D≤n)
    moreover from A1 A3 have D∈ℤ  ∀n∈A. n∈ℤ
      using Nonnegative_def by auto
    ultimately have ∀n∈A. n≤D
      using Int_ZF_2_L19 by blast
    hence ∀n∈A. ⟨n,D⟩ ∈ IntegerOrder by simp
    then have IsBoundedAbove(A,IntegerOrder)
      by (rule Order_ZF_3_L10)
    with A1 A2 have False using Int_ZF_1_5_L1A IsBounded_def
      Int_bounded_iff_fin by auto
  } thus thesis by auto
qed
```

Infinite subsets of Z_+ do not have a maximum - If $A \subseteq Z_+$ then for every integer we can find one in the set that is not smaller. This is very similar to Int_ZF_1_5_L2, except we have $\mathbb{Z}_+$ instead of $\mathbb{Z}^+$ here.

```
lemma (in int0) Int_ZF_1_5_L2A:
  assumes A1: A ⊆ ℤ₊  and A2: A ∉ Fin(ℤ) and A3: D∈ℤ
  shows ∃n∈A. D≤n
proof -
{ assume ∀n∈A. ¬(D≤n)
    moreover from A1 A3 have D∈ℤ  ∀n∈A. n∈ℤ
      using PositiveSet_def by auto
    ultimately have ∀n∈A. n≤D
      using Int_ZF_2_L19 by blast
    hence ∀n∈A. ⟨n,D⟩ ∈ IntegerOrder by simp
    then have IsBoundedAbove(A,IntegerOrder)
      by (rule Order_ZF_3_L10)
    with A1 A2 have False using Int_ZF_1_5_L1B IsBounded_def
      Int_bounded_iff_fin by auto
  } thus thesis by auto
qed
```

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

```
lemma (in int0) Int_decomp: assumes m∈ℤ
  shows Exactly_1_of_3_holds (m=0,m∈ℤ₊,(-m)∈ℤ₊)
  using prems Int_ZF_2_T1 group3.OrdGroup_decomp
  by simp
```

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

```
lemma (in int0) int_decomp_cases: assumes m∈ℤ
  shows m=0 ∨ m∈ℤ₊ ∨ (-m) ∈ ℤ₊
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L14
  by simp
```

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

```
lemma (in int0) Int_ZF_1_5_L3: shows m∈ℤ₊ ⟷ 1≤m
proof
  assume m∈ℤ₊ then have 0≤m  m≠0
    using PositiveSet_def by auto
  then have 0+1 ≤ m
    using Int_ZF_4_L1B by auto
  then show 1≤m
    using int_zero_one_are_int Int_ZF_1_T2 group0.group0_2_L2
    by simp
next assume 1≤m
  then have m∈ℤ  0≤m  m≠0
    using Int_ZF_2_L1A Int_ZF_2_L16C by auto
  then show m∈ℤ₊ using PositiveSet_def by auto
qed
```

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

```
lemma (in int0) pos_int_closed_mul_unfold:
  assumes a∈ℤ₊  b∈ℤ₊
  shows a·b ∈ ℤ₊
  using prems Int_ZF_1_5_L3 Int_ZF_1_3_L3 by simp
```

The set of positive integers is closed under multiplication.

```
lemma (in int0) pos_int_closed_mul: shows
  ℤ₊ {is closed under} IntegerMultiplication
  using pos_int_closed_mul_unfold IsOpClosed_def
  by simp
```

It is an overkill to prove that the ring of integers has no zero divisors this way, but why not?

```
lemma (in int0) int_has_no_zero_divs:
  shows HasNoZeroDivs(ℤ,IntegerAddition,IntegerMultiplication)
  using pos_int_closed_mul Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L3
  by simp
```

Nonnegative integers are positive ones plus zero.

```
lemma (in int0) Int_ZF_1_5_L3A: shows ℤ⁺ = ℤ₊ ∪ {0}
  using Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L24 by simp
```

We can make a function smaller than any constant on a given interval of positive integers by adding another constant.

```
lemma (in int0) Int_ZF_1_5_L4:
  assumes A1: f:ℤ→ℤ and A2: K∈ℤ N∈ℤ
  shows ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
proof -
  from A2 have N≤1 ∨ 2≤N
    using int_zero_one_are_int no_int_between
    by simp
  moreover
  { assume A3: N≤1
    let C = 0
    have C ∈ ℤ using int_zero_one_are_int
      by simp
    moreover
    { fix n assume n∈ℤ₊
      then have 1 ≤ n using Int_ZF_1_5_L3
        by simp
      with A3 have N≤n by (rule Int_order_transitive)
    } then have  ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
      by auto
    ultimately have ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
      by auto }
  moreover
```

```
  { let C = K - 1 - maxf(f,1..(N-1))
    assume 2≤N
    then have 2-1 ≤ N-1
      using int_zero_one_are_int Int_ZF_1_1_L4 int_ord_transl_inv
      by simp
    then have I: 1 ≤ N-1
      using int_zero_one_are_int Int_ZF_1_2_L3 by simp
    with A1 A2 have T:
      maxf(f,1..(N-1)) ∈ ℤ  K-1 ∈ ℤ  C ∈ ℤ
      using Int_ZF_1_4_L2 Int_ZF_1_1_L5 int_zero_one_are_int
      by auto
    moreover
    { fix n assume A4: n∈ℤ+
      { assume A5: K ≤ f(n) + C and ¬(N≤n)
        with A2 A4 have n ≤ N-1
          using PositiveSet_def Int_ZF_1_3_L6A by simp
        with A4 have n ∈ 1..(N-1)
          using Int_ZF_1_5_L3 Interval_def by auto
        with A1 I T have f(n)+C ≤ maxf(f,1..(N-1)) + C
          using Int_ZF_1_4_L2 int_ord_transl_inv by simp
        with T have f(n)+C ≤ K-1
          using Int_ZF_1_2_L3 by simp
        with A5 have K ≤  K-1
          by (rule Int_order_transitive)
        with A2 have False using Int_ZF_1_2_L3AA by simp
      } then have K ≤ f(n) + C ⟶ N≤n
        by auto
    } then have ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
      by simp
    ultimately have ∃C∈ℤ. ∀n∈ℤ+. K ≤ f(n) + C ⟶ N≤n
      by auto }
  ultimately show thesis by auto
qed
```

Absolute value is identity on positive integers.

```
lemma (in int0) Int_ZF_1_5_L4A:
  assumes a∈ℤ+ shows abs(a) = a
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_3_L2B
  by simp
```

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

```
lemma (in int0) int_one_two_are_pos: shows 1 ∈ ℤ+  2 ∈ ℤ+
  using int_zero_one_are_int int_ord_is_refl refl_def Int_ZF_1_5_L3
  Int_ZF_2_L16B by auto
```

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

```
lemma (in int0) Int_ZF_1_5_L5: assumes A1: f : ℤ→X
  shows f(ℤ+) ≠ 0
proof -
```

```
  have ℤ₊ ⊆ ℤ using PositiveSet_def by auto
  with A1 show f(ℤ₊) ≠ 0
    using int_one_two_are_pos func_imagedef by auto
qed
```

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

```
lemma (in int0) Int_ZF_1_5_L6: assumes A1: n ∈ ℤ₊
  shows
  0 ≤ n-1
  0 ∈ 0..(n-1)
  0..(n-1) ⊆ ℤ
proof -
  from A1 have 1 ≤ n  (-1) ∈ ℤ
    using Int_ZF_1_5_L3 int_zero_one_are_int Int_ZF_1_1_L4
    by auto
  then have 1-1 ≤ n-1
    using int_ord_transl_inv by simp
  then show 0 ≤ n-1
    using int_zero_one_are_int Int_ZF_1_1_L4 by simp
  then show 0 ∈ 0..(n-1)
    using int_zero_one_are_int int_ord_is_refl refl_def Order_ZF_2_L1B
    by simp
  show 0..(n-1) ⊆ ℤ
    using Int_ZF_2_L1B Order_ZF_2_L6 by simp
qed
```

Intgers greater than one in $\mathbb{Z}_+$ belong to $\mathbb{Z}_+$. This is a property of ordered groups and follows from OrderedGroup_ZF_1_L19, but Isabelle's simplifier has problems using that result directly, so we reprove it specifically for integers.

```
lemma (in int0) Int_ZF_1_5_L7:  assumes a ∈ ℤ₊ and a≤b
  shows b ∈ ℤ₊
proof-
  from prems have 1≤a  a≤b
    using Int_ZF_1_5_L3 by auto
  then have 1≤b by (rule Int_order_transitive)
  then show b ∈ ℤ₊ using Int_ZF_1_5_L3 by simp
qed
```

Adding a positive integer increases integers.

```
lemma (in int0) Int_ZF_1_5_L7A: assumes a∈ℤ  b ∈ ℤ₊
  shows a ≤ a+b  a ≠ a+b  a+b ∈ ℤ
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L22
  by auto
```

For any integer m the greater of m and 1 is a positive integer that is greater or equal than m. If we add 1 to it we get a positive integer that is strictly greater than m.

```
lemma (in int0) Int_ZF_1_5_L7B: assumes a∈ℤ
```

```
  shows
  a ≤ GreaterOf(IntegerOrder,1,a)
  GreaterOf(IntegerOrder,1,a) ∈ ℤ+
  GreaterOf(IntegerOrder,1,a) + 1 ∈ ℤ+
  a ≤ GreaterOf(IntegerOrder,1,a) + 1
  a ≠ GreaterOf(IntegerOrder,1,a) + 1
  using prems int_zero_not_one Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L12
  by auto
```

The opposite of an element of $\mathbb{Z}_+$ cannot belong to $\mathbb{Z}_+$.

```
lemma (in int0) Int_ZF_1_5_L8: assumes a ∈ ℤ+
  shows (-a) ∉ ℤ+
  using prems Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L20
  by simp
```

For every integer there is one in $\mathbb{Z}_+$ that is greater or equal.

```
lemma (in int0) Int_ZF_1_5_L9: assumes a∈ℤ
  shows ∃b∈ℤ+. a≤b
  using prems int_not_trivial Int_ZF_2_T1 group3.OrderedGroup_ZF_1_L23
  by simp
```

A theorem about odd extensions. Recall from `OrdereGroup_ZF.thy` that the odd extension of an integer function f defined on $\mathbb{Z}_+$ is the odd function on $\mathbb{Z}$ equal to f on $\mathbb{Z}_+$. First we show that the odd extension is defined on $\mathbb{Z}$.

```
lemma (in int0) Int_ZF_1_5_L10: assumes f : ℤ+→ℤ
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,f) : ℤ→ℤ
  using prems Int_ZF_2_T1 group3.odd_ext_props by simp
```

On $\mathbb{Z}_+$, the odd extension of f is the same as f.

```
lemma (in int0) Int_ZF_1_5_L11: assumes f : ℤ+→ℤ and a ∈ ℤ+ and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows g(a) = f(a)
  using prems Int_ZF_2_T1 group3.odd_ext_props by simp
```

On -$\mathbb{Z}_+$, the value of the odd extension of f is the negative of $f(-a)$.

```
lemma (in int0) Int_ZF_1_5_L12:
  assumes f : ℤ+→ℤ and a ∈ (-ℤ+) and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows g(a) = -(f(-a))
  using prems Int_ZF_2_T1 group3.odd_ext_props by simp
```

Odd extensions are odd on $\mathbb{Z}$.

```
lemma (in int0) int_oddext_is_odd:
  assumes f : ℤ+→ℤ and a∈ℤ and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows g(-a) = -(g(a))
  using prems Int_ZF_2_T1 group3.oddext_is_odd by simp
```

Alternative definition of an odd function.

```
lemma (in int0) Int_ZF_1_5_L13:  assumes A1: f: ℤ→ℤ shows
  (∀a∈ℤ. f(-a) = (-f(a))) ⟷ (∀a∈ℤ. (-(f(-a))) = f(a))
  using prems Int_ZF_1_T2 group0.group0_6_L2 by simp
```

Another way of expressing the fact that odd extensions are odd.

```
lemma (in int0) int_oddext_is_odd_alt:
  assumes f : ℤ+→ℤ and a∈ℤ and
  g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  shows (-g(-a)) = g(a)
  using prems Int_ZF_2_T1 group3.oddext_is_odd_alt by simp
```

24.6 Functions with infinite limits

In this section we consider functions (integer sequences) that have infinite limits. An integer function has infinite positive limit if it is arbitrarily large for large enough arguments. Similarly, a function has infinite negative limit if it is arbitrarily small for small enough arguments. The material in this come mostly from the section in OrderedGroup_ZF.thy with he same title. Here we rewrite the theorems from that section in the notation we use for integers and add some results specific for the ordered group of integers.

If an image of a set by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in int0) Int_ZF_1_6_L1: assumes f: ℤ→ℤ and
  ∀a∈ℤ.∃b∈ℤ+.∀x. b≤x ⟶ a ≤ f(x) and A ⊆ ℤ and
  IsBoundedAbove(f(A),IntegerOrder)
  shows IsBoundedAbove(A,IntegerOrder)
  using prems int_not_trivial Int_ZF_2_T1 group3.OrderedGroup_ZF_7_L1
  by simp
```

If an image of a set defined by separation by a function with infinite positive limit is bounded above, then the set itself is bounded above.

```
lemma (in int0) Int_ZF_1_6_L2: assumes A1: X≠0 and A2: f: ℤ→ℤ and

  A3: ∀a∈ℤ.∃b∈ℤ+.∀x. b≤x ⟶ a ≤ f(x) and
  A4: ∀x∈X. b(x) ∈ ℤ  ∧ f(b(x)) ≤ U
  shows ∃u.∀x∈X. b(x) ≤ u
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 have
    group3(G, P, r)
    r {is total on} G
    G ≠ {TheNeutralElement(G, P)}
    X≠0  f: G→G
```

```
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y. ⟨b, y⟩ ∈ r ⟶ ⟨a, f(y)⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨f(b(x)), U⟩ ∈ r
    using int_not_trivial Int_ZF_2_T1 by auto
  then have ∃u. ∀x∈X. ⟨b(x), u⟩ ∈ r by (rule group3.OrderedGroup_ZF_7_L2)
  thus thesis by simp
qed
```

If an image of a set defined by separation by a integer function with infinite negative limit is bounded below, then the set itself is bounded above. This is dual to `Int_ZF_1_6_L2`.

```
lemma (in int0) Int_ZF_1_6_L3: assumes A1: X≠0 and A2: f: ℤ→ℤ and

  A3: ∀a∈ℤ.∃b∈ℤ₊.∀y. b≤y ⟶ f(-y) ≤ a and
  A4: ∀x∈X. b(x) ∈ ℤ  ∧ L ≤ f(b(x))
  shows ∃l.∀x∈X. l ≤ b(x)
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 have
    group3(G, P, r)
    r {is total on} G
    G ≠ {TheNeutralElement(G, P)}
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y.
    ⟨b, y⟩ ∈ r ⟶ ⟨f(GroupInv(G, P)(y)),a⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨L,f(b(x))⟩ ∈ r
    using int_not_trivial Int_ZF_2_T1 by auto
  then have ∃l. ∀x∈X. ⟨l, b(x)⟩ ∈ r by (rule group3.OrderedGroup_ZF_7_L3)
  thus thesis by simp
qed
```

The next lemma combines `Int_ZF_1_6_L2` and `Int_ZF_1_6_L3` to show that if the image of a set defined by separation by a function with infinite limits is bounded, then the set itself is bounded. The proof again uses directly a fact from `OrderedGroup_ZF.thy`.

```
lemma (in int0) Int_ZF_1_6_L4:
  assumes A1: X≠0 and A2: f: ℤ→ℤ and
  A3: ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x) and
  A4: ∀a∈ℤ.∃b∈ℤ₊.∀y. b≤y ⟶ f(-y) ≤ a and
  A5: ∀x∈X. b(x) ∈ ℤ ∧ f(b(x)) ≤ U ∧ L ≤ f(b(x))
  shows ∃M.∀x∈X. abs(b(x)) ≤ M
proof -
  let G = ℤ
  let P = IntegerAddition
  let r = IntegerOrder
  from A1 A2 A3 A4 A5 have
    group3(G, P, r)
```

```
    r {is total on} G
    G ≠ {TheNeutralElement(G, P)}
    X≠0  f: G→G
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y. ⟨b, y⟩ ∈ r ⟶ ⟨a, f(y)⟩ ∈ r
    ∀a∈G. ∃b∈PositiveSet(G, P, r). ∀y.
    ⟨b, y⟩ ∈ r ⟶ ⟨f(GroupInv(G, P)(y)),a⟩ ∈ r
    ∀x∈X. b(x) ∈ G ∧ ⟨L,f(b(x))⟩ ∈ r ∧ ⟨f(b(x)), U⟩ ∈ r
    using int_not_trivial Int_ZF_2_T1 by auto
  then have ∃M. ∀x∈X. ⟨AbsoluteValue(G, P, r)  b(x), M⟩ ∈ r
    by (rule group3.OrderedGroup_ZF_7_L4)
  thus thesis by simp
qed
```

If a function is larger than some constant for arguments large enough, then the image of a set that is bounded below is bounded below. This is not true for ordered groups in general, but only for those for which bounded sets are finite. This does not require the function to have infinite limit, but such functions do have this property.

```
lemma (in int0) Int_ZF_1_6_L5:
  assumes A1: f: ℤ→ℤ and A2: N∈ℤ and
  A3: ∀m. N≤m ⟶ L ≤ f(m) and
  A4: IsBoundedBelow(A,IntegerOrder)
  shows IsBoundedBelow(f(A),IntegerOrder)
proof -
  from A2 A4 have A = {x∈A. x≤N} ∪ {x∈A. N≤x}
    using Int_ZF_2_T1 Int_ZF_2_L1C Order_ZF_1_L5
    by simp
  moreover have
    f({x∈A. x≤N} ∪ {x∈A. N≤x}) =
    f{x∈A. x≤N} ∪ f{x∈A. N≤x}
    by (rule image_Un)
  ultimately have f(A) = f{x∈A. x≤N} ∪ f{x∈A. N≤x}
    by simp
  moreover have IsBoundedBelow(f{x∈A. x≤N},IntegerOrder)
  proof -
    let B = {x∈A. x≤N}
    from A4 have B ∈ Fin(ℤ)
      using Order_ZF_3_L16 Int_bounded_iff_fin by auto
    with A1 have  IsBounded(f(B),IntegerOrder)
      using Finite1_L6A Int_bounded_iff_fin by simp
    then show IsBoundedBelow(f(B),IntegerOrder)
      using IsBounded_def by simp
  qed
  moreover have IsBoundedBelow(f{x∈A. N≤x},IntegerOrder)
  proof -
    let C = {x∈A. N≤x}
    from A4 have C ⊆ ℤ using Int_ZF_2_L1C by auto
    with A1 A3 have ∀y ∈ f(C). ⟨L,y⟩ ∈ IntegerOrder
      using func_imagedef by simp
```

```
    then show IsBoundedBelow(f(C),IntegerOrder)
      by (rule Order_ZF_3_L9)
  qed
  ultimately show IsBoundedBelow(f(A),IntegerOrder)
    using Int_ZF_2_T1 Int_ZF_2_L6 Int_ZF_2_L1B Order_ZF_3_L6
    by simp
qed
```

A function that has an infinite limit can be made arbitrarily large on positive integers by adding a constant. This does not actually require the function to have infinite limit, just to be larger than a constant for arguments large enough.

```
lemma (in int0) Int_ZF_1_6_L6: assumes A1: N∈ℤ and
  A2: ∀m. N≤m ⟶ L ≤ f(m) and
  A3: f: ℤ→ℤ and A4: K∈ℤ
  shows ∃c∈ℤ. ∀n∈ℤ₊. K ≤ f(n)+c
proof -
  have IsBoundedBelow(ℤ₊,IntegerOrder)
    using Int_ZF_1_5_L1 by simp
  with A3 A1 A2 have IsBoundedBelow(f(ℤ₊),IntegerOrder)
    by (rule Int_ZF_1_6_L5)
  with A1 obtain l where I: ∀y∈f(ℤ₊). l ≤ y
    using Int_ZF_1_5_L5 IsBoundedBelow_def by auto
  let c = K-l
  from A3 have f(ℤ₊) ≠ 0 using Int_ZF_1_5_L5
    by simp
  then have ∃y. y ∈ f(ℤ₊) by (rule nonempty_has_element)
  then obtain y where y ∈ f(ℤ₊) by auto
  with A4 I have T: l ∈ ℤ  c ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  { fix n assume A5: n∈ℤ₊
    have ℤ₊ ⊆ ℤ using PositiveSet_def by auto
    with A3 I T A5 have l + c ≤ f(n) + c
      using func_imagedef int_ord_transl_inv by auto
     with I T have l + c ≤ f(n) + c
      using int_ord_transl_inv by simp
    with A4 T have K ≤  f(n) + c
      using Int_ZF_1_2_L3 by simp
  } then have ∀n∈ℤ₊. K ≤  f(n) + c by simp
  with T show thesis by auto
qed
```

If a function has infinite limit, then we can add such constant such that minimum of those arguments for which the function (plus the constant) is larger than another given constant is greater than a third constant. It is not as complicated as it sounds.

```
lemma (in int0) Int_ZF_1_6_L7:
  assumes A1: f: ℤ→ℤ and A2: K∈ℤ  N∈ℤ and
```

```
  A3: ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x)
  shows ∃C∈ℤ. N ≤ Minimum(IntegerOrder,{n∈ℤ₊. K ≤ f(n)+C})
proof -
  from A1 A2 have ∃C∈ℤ. ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
    using Int_ZF_1_5_L4 by simp
  then obtain C where I: C∈ℤ and
    II: ∀n∈ℤ₊. K ≤ f(n) + C ⟶ N≤n
    by auto
  have antisym(IntegerOrder) using Int_ZF_2_L4 by simp
  moreover have HasAminimum(IntegerOrder,{n∈ℤ₊. K ≤ f(n)+C})
  proof -
    from A2 A3 I have ∃n∈ℤ₊.∀x. n≤x ⟶ K-C ≤ f(x)
      using Int_ZF_1_1_L5 by simp
    then obtain n where
      n∈ℤ₊ and ∀x. n≤x ⟶ K-C ≤  f(x)
      by auto
    with A2 I have
      {n∈ℤ₊. K ≤ f(n)+C} ≠ 0
      {n∈ℤ₊. K ≤ f(n)+C} ⊆ ℤ₊
      using int_ord_is_refl refl_def PositiveSet_def Int_ZF_2_L9C
      by auto
    then show HasAminimum(IntegerOrder,{n∈ℤ₊. K ≤ f(n)+C})
      using Int_ZF_1_5_L1C by simp
  qed
  moreover from II have
    ∀n ∈ {n∈ℤ₊. K ≤ f(n)+C}. ⟨N,n⟩ ∈ IntegerOrder
    by auto
  ultimately have
    ⟨N,Minimum(IntegerOrder,{n∈ℤ₊. K ≤ f(n)+C})⟩ ∈ IntegerOrder
    by (rule Order_ZF_4_L12)
  with I show thesis by auto
qed
```

For any integer m the function $k \mapsto m \cdot k$ has an infinite limit (or negative of that). This is why we put some properties of these functions here, even though they properly belong to a (yet nonexistent) section on homomorphisms. The next lemma shows that the set $\{a \cdot x : x \in Z\}$ can finite only if $a = 0$.

```
lemma (in int0) Int_ZF_1_6_L8:
  assumes A1: a∈ℤ and A2: {a·x. x∈ℤ} ∈ Fin(ℤ)
  shows a = 0
proof -
  from A1 have a=0 ∨ (a ≤ -1) ∨ (1≤a)
    using Int_ZF_1_3_L6C by simp
  moreover
  { assume a ≤ -1
    then have {a·x. x∈ℤ} ∉ Fin(ℤ)
      using int_zero_not_one Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L6
      by simp
```

```
    with A2 have False by simp }
  moreover
  { assume 1≤a
    then have {a·x. x∈ℤ} ∉ Fin(ℤ)
      using int_zero_not_one Int_ZF_1_3_T1 ring1.OrdRing_ZF_3_L5
    by simp
  with A2 have False by simp }
  ultimately show  a = 0 by auto
qed
```

24.7 Miscelaneous

In this section we put some technical lemmas needed in various other places that are hard to classify.

Suppose we have an integer expression (a meta-function)F such that $F(p)|p|$ is bounded by a linear function of $|p|$, that is for some integers A, B we have $F(p)|p| \leq A|p| + B$. We show that F is then bounded. The proof is easy, we just divide both sides by $|p|$ and take the limit (just kidding).

```
lemma (in int0) Int_ZF_1_7_L1:
  assumes A1: ∀q∈ℤ. F(q) ∈ ℤ and
  A2:  ∀q∈ℤ. F(q)·abs(q) ≤ A·abs(q) + B and
  A3: A∈ℤ  B∈ℤ
  shows ∃L. ∀p∈ℤ. F(p) ≤ L
proof -
  let I = (-abs(B))..abs(B)
  def DefK: K == {F(q). q ∈ I}
  let M = Maximum(IntegerOrder,K)
  let L = GreaterOf(IntegerOrder,M,A+1)
  from A3 A1 DefK have C1:
    IsBounded(I,IntegerOrder)
    I ≠ 0
    ∀q∈ℤ. F(q) ∈ ℤ
    K = {F(q). q ∈ I}
    using Order_ZF_3_L11 Int_ZF_1_3_L17 by auto
  then have M ∈ ℤ by (rule Int_ZF_1_4_L1)
  with A3 have T1: M ≤ L  A+1 ≤ L
    using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_1_3_L18
    by auto
  from C1 have T2: ∀q∈I. F(q) ≤ M
    by (rule Int_ZF_1_4_L1)
  { fix p assume A4: p∈ℤ have F(p) ≤ L
    proof (cases abs(p) ≤ abs(B))
      assume abs(p) ≤ abs(B)
      with A4 T1 T2 have F(p) ≤ M  M ≤ L
        using Int_ZF_1_3_L19 by auto
      then show F(p) ≤ L by (rule Int_order_transitive)
    next assume A5: ¬(abs(p) ≤ abs(B))
      from A3 A2 A4 have
```

```
      A·abs(p) ∈ ℤ  F(p)·abs(p) ≤ A·abs(p) + B
      using Int_ZF_2_L14 Int_ZF_1_1_L5 by auto
    moreover from A3 A4 A5 have B ≤ abs(p)
      using Int_ZF_1_3_L15 by simp
    ultimately have
      F(p)·abs(p) ≤ A·abs(p) + abs(p)
      using Int_ZF_2_L15A by blast
    with A3 A4 have F(p)·abs(p) ≤ (A+1)·abs(p)
      using Int_ZF_2_L14 Int_ZF_1_2_L7 by simp
    moreover from A3 A1 A4 A5 have
      F(p) ∈ ℤ  A+1 ∈ ℤ  abs(p) ∈ ℤ
       ¬(abs(p) ≤ 0)
      using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_2_L14 Int_ZF_1_3_L11
      by auto
    ultimately have F(p) ≤ A+1
      using Int_ineq_simpl_positive by simp
    moreover from T1 have  A+1 ≤ L by simp
    ultimately show F(p) ≤ L by (rule Int_order_transitive)
  qed
 } then have ∀p∈ℤ. F(p) ≤ L by simp
 thus thesis by auto
qed
```

A lemma about splitting (not really, there is some overlap) the 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
  using prems Int_ZF_2_T1 group3.OrdGroup_6cases by simp

end
```

25 IntDiv_ZF.thy

```
theory IntDiv_ZF imports Int_ZF_1 IntDiv

begin
```

This theory translates some results form the Isabelle's `IntDiv.thy` theory to the notation used by IsarMathLib.

25.1 Quotient and reminder

For any integers m, n , $n > 0$ there are unique integers q, p such that $0 \leq p < n$ and $m = n \cdot q + p$. Number p in this decompsition is usually called m mod n. Standard Isabelle denotes numbers q, p as `m zdiv n` and `m zmod n`, resp., and we will use the same notation.

The next lemma is sometimes called the "quotient-reminder theorem".

```
lemma (in int0) IntDiv_ZF_1_L1: assumes m∈ℤ  n∈ℤ
  shows m = n·(m zdiv n) + (m zmod n)
  using prems Int_ZF_1_L2 raw_zmod_zdiv_equality
  by simp
```

If $n > 0$ then `m zmod n` is between 0 and $n - 1$.

```
lemma (in int0) IntDiv_ZF_1_L2:
  assumes A1: m∈ℤ and A2: 0≤n  n≠0
  shows
  0 ≤ m zmod n
  m zmod n ≤ n    m zmod n ≠ n
  m zmod n ≤ n-1
proof -
  from A2 have T: n ∈ ℤ
    using Int_ZF_2_L1A by simp
  from A2 have #0 $< n using Int_ZF_2_L9 Int_ZF_1_L8
    by auto
  with T show
    0 ≤ m zmod n
    m zmod n ≤ n
    m zmod n ≠ n
    using pos_mod Int_ZF_1_L8 Int_ZF_1_L8A zmod_type
      Int_ZF_2_L1 Int_ZF_2_L9AA
    by auto
  then show m zmod n ≤ n-1
    using Int_ZF_4_L1B by auto
qed
```

$(m \cdot k)$ div $k = m$.

```
lemma (in int0) IntDiv_ZF_1_L3:
  assumes m∈ℤ  k∈ℤ  and k≠0
```

```
  shows
  (m·k) zdiv k = m
  (k·m) zdiv k = m
  using prems zdiv_zmult_self1 zdiv_zmult_self2
    Int_ZF_1_L8 Int_ZF_1_L2 by auto
```

The next lemma essentially translates zdiv_mono1 from standard Isabelle to our notation.

```
lemma (in int0) IntDiv_ZF_1_L4:
  assumes A1: m ≤ k and A2: 0≤n  n≠0
  shows m zdiv n ≤  k zdiv n
proof -
  from A2 have #0 ≤ n  #0 ≠ n
    using Int_ZF_1_L8 by auto
  with A1 have
    m zdiv n $≤ k zdiv n
    m zdiv n ∈ ℤ    m zdiv k ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_2_L9 zdiv_mono1
    by auto
  then show (m zdiv n) ≤ (k zdiv n)
    using Int_ZF_2_L1 by simp
qed
```

A quotient-reminder theorem about integers greater than a given product.

```
lemma (in int0) IntDiv_ZF_1_L5:
  assumes A1: n ∈ ℤ+ and A2: n ≤ k and A3: k·n ≤ m
  shows
  m = n·(m zdiv n) + (m zmod n)
  m = (m zdiv n)·n + (m zmod n)
  (m zmod n) ∈ 0..(n-1)
  k ≤ (m zdiv n)
  m zdiv n ∈ ℤ+
proof -
  from A2 A3 have T:
    m∈ℤ  n∈ℤ  k∈ℤ  m zdiv n ∈ ℤ
    using Int_ZF_2_L1A by auto
   then show m = n·(m zdiv n) + (m zmod n)
     using IntDiv_ZF_1_L1 by simp
   with T show m = (m zdiv n)·n + (m zmod n)
     using Int_ZF_1_L4 by simp
    from A1 have I: 0≤n  n≠0
     using PositiveSet_def by auto
   with T show (m zmod n) ∈ 0..(n-1)
    using IntDiv_ZF_1_L2 Order_ZF_2_L1
    by simp
  from A3 I have (k·n zdiv n) ≤ (m zdiv n)
    using IntDiv_ZF_1_L4 by simp
  with I T show k ≤ (m zdiv n)
    using IntDiv_ZF_1_L3 by simp
```

```
  with A1 A2 show m zdiv n ∈ ℤ₊
    using Int_ZF_1_5_L7 by blast
qed

end
```

26 Int_ZF_2.thy

theory `Int_ZF_2` **imports** `Int_ZF_1 IntDiv_ZF 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_x.thy` series.

26.1 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.thy` 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.thy` 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 :: [i,i]⇒o (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 :: i⇒i (-_ [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)
  using Int_ZF_1_T2 group1_axioms.intro group1_def by simp
```

Type information related to the homomorphism difference expression.

```
lemma (in int1) Int_ZF_2_1_L2: assumes f∈𝒮 and n∈ℤ m∈ℤ
  shows
  m+n ∈ ℤ
  f(m+n) ∈ ℤ
  f(m) ∈ ℤ   f(n) ∈ ℤ
  f(m) + f(n) ∈ ℤ
  HomDiff(ℤ,IntegerAddition,f,<m,n>) ∈ ℤ
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_2_L4A
  by auto
```

Type information related to the homomorphism difference expression.

```
lemma (in int1) Int_ZF_2_1_L2A:
  assumes f:ℤ→ℤ and n∈ℤ  m∈ℤ
  shows
  m+n ∈ ℤ
  f(m+n) ∈ ℤ   f(m) ∈ ℤ   f(n) ∈ ℤ
  f(m) + f(n) ∈ ℤ
  HomDiff(ℤ,IntegerAddition,f,<m,n>) ∈ ℤ
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_2_L4
  by auto
```

Slopes map integers into integers.

```
lemma (in int1) Int_ZF_2_1_L2B:
  assumes A1: f∈𝒮 and A2: m∈ℤ
  shows f(m) ∈ ℤ
proof -
  from A1 have f:ℤ→ℤ using AlmostHoms_def by simp
  with A2 show f(m) ∈ ℤ using apply_funtype by simp
qed
```

The homomorphism difference in multiplicative notation is defined as the expression $s(m \cdot n) \cdot (s(m) \cdot s(n))^{-1}$. The next lemma shows that in the additive notation used for integers the homomorphism difference is $f(m+n) - f(m) - f(n)$ which we denote as δ(f,m,n).

```
lemma (in int1) Int_ZF_2_1_L3:
  assumes f:ℤ→ℤ and m∈ℤ  n∈ℤ
  shows HomDiff(ℤ,IntegerAddition,f,<m,n>) = δ(f,m,n)
  using prems Int_ZF_2_1_L2A Int_ZF_1_T2 group0.group0_4_L4A
    HomDiff_def by auto
```

The next formula restates the definition of the homomorphism difference to express the value an almost homomorphism on a sum.

```
lemma (in int1) Int_ZF_2_1_L3A:
  assumes A1: f∈𝒮 and A2: m∈ℤ  n∈ℤ
  shows
  f(m+n) = f(m)+(f(n)+δ(f,m,n))
proof -
  from A1 A2 have
    T: f(m)∈ ℤ  f(n) ∈ ℤ  δ(f,m,n) ∈ ℤ and
    HomDiff(ℤ,IntegerAddition,f,<m,n>) = δ(f,m,n)
    using Int_ZF_2_1_L2 AlmostHoms_def Int_ZF_2_1_L3 by auto
  with A1 A2 show  f(m+n) = f(m)+(f(n)+δ(f,m,n))
    using Int_ZF_2_1_L3 Int_ZF_1_L3
      Int_ZF_2_1_L1 group1.Group_ZF_3_4_L1
    by simp
qed
```

The homomorphism difference of any integer function is integer.

```
lemma (in int1) Int_ZF_2_1_L3B:
  assumes f:ℤ→ℤ and m∈ℤ  n∈ℤ
  shows δ(f,m,n) ∈ ℤ
  using prems Int_ZF_2_1_L2A Int_ZF_2_1_L3 by simp
```

The value of an integer function at a sum expressed in terms of δ.

```
lemma (in int1) Int_ZF_2_1_L3C: assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows f(m+n) = δ(f,m,n) + f(n) + f(m)
proof -
  from A1 A2 have T:
    δ(f,m,n) ∈ ℤ  f(m+n) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ
    using Int_ZF_1_1_L5 apply_funtype by auto
  then show f(m+n) = δ(f,m,n) + f(n) + f(m)
    using Int_ZF_1_2_L15 by simp
qed
```

The next lemma presents two ways the set of homomorphism differences can be written.

```
lemma (in int1) Int_ZF_2_1_L4: assumes A1: f:ℤ→ℤ
  shows {abs(HomDiff(ℤ,IntegerAddition,f,x)). x ∈ ℤ×ℤ} =
  {abs(δ(f,m,n)). <m,n> ∈ ℤ×ℤ}
proof -
  from A1 have ∀m∈ℤ. ∀n∈ℤ.
    abs(HomDiff(ℤ,IntegerAddition,f,<m,n>)) = abs(δ(f,m,n))
    using Int_ZF_2_1_L3 by simp
  then show thesis by (rule ZF1_1_L4A)
qed
```

If f maps integers into integers and for all $m,n \in Z$ we have $|f(m+n) - f(m) - f(n)| \leq L$ for some L, then f is a slope.

```
lemma (in int1) Int_ZF_2_1_L5: assumes A1: f:ℤ→ℤ
  and A2: ∀m∈ℤ.∀n∈ℤ. abs(δ(f,m,n)) ≤ L
```

```
  shows f∈𝒮
proof -
  let Abs = AbsoluteValue(ℤ,IntegerAddition,IntegerOrder)
  have group3(ℤ,IntegerAddition,IntegerOrder)
    IntegerOrder {is total on} ℤ
    using Int_ZF_2_T1 by auto
  moreover from A1 A2 have
    ∀x∈ℤ×ℤ. HomDiff(ℤ,IntegerAddition,f,x) ∈ ℤ ∧
    ⟨Abs(HomDiff(ℤ,IntegerAddition,f,x)),L ⟩ ∈ IntegerOrder
    using Int_ZF_2_1_L2A Int_ZF_2_1_L3 by auto
  ultimately have
    IsBounded({HomDiff(ℤ,IntegerAddition,f,x). x∈ℤ×ℤ},IntegerOrder)
    by (rule group3.OrderedGroup_ZF_3_L9A)
  with A1 show f ∈ 𝒮 using Int_bounded_iff_fin AlmostHoms_def
    by simp
qed
```

The absolute value of homomorphism difference of a slope s does not exceed maxδ(s).

```
lemma (in int1) Int_ZF_2_1_L7:
  assumes A1: s∈𝒮 and A2: n∈ℤ  m∈ℤ
  shows
  abs(δ(s,m,n)) ≤ maxδ(s)
  δ(s,m,n) ∈ ℤ   maxδ(s) ∈ ℤ
  (-maxδ(s)) ≤ δ(s,m,n)
proof -
  from A1 A2 show T: δ(s,m,n) ∈ ℤ
    using Int_ZF_2_1_L2 Int_ZF_1_1_L5 by simp
  let A = {abs(HomDiff(ℤ,IntegerAddition,s,x)). x∈ℤ×ℤ}
  let B = {abs(δ(s,m,n)). <m,n> ∈ ℤ×ℤ}
  let d = abs(δ(s,m,n))
  have IsLinOrder(ℤ,IntegerOrder) using Int_ZF_2_T1
    by simp
  moreover have A ∈ Fin(ℤ)
  proof -
    have ∀k∈ℤ. abs(k) ∈ ℤ using Int_ZF_2_L14 by simp
    moreover from A1 have
      {HomDiff(ℤ,IntegerAddition,s,x). x ∈ ℤ×ℤ} ∈ Fin(ℤ)
      using AlmostHoms_def by simp
    ultimately show A ∈ Fin(ℤ) by (rule Finite1_L6C)
  qed
  moreover have A≠0 by auto
  ultimately have ∀k∈A. ⟨k,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule Finite_ZF_1_T2)
  moreover from A1 A2 have d∈A using AlmostHoms_def Int_ZF_2_1_L4
    by auto
  ultimately have d ≤ Maximum(IntegerOrder,A) by auto
  with A1 show d ≤ maxδ(s)  maxδ(s) ∈ ℤ
    using AlmostHoms_def Int_ZF_2_1_L4 Int_ZF_2_L1A
```

```
    by auto
  with T show (-maxδ(s)) ≤ δ(s,m,n)
    using Int_ZF_1_3_L19 by simp
qed
```

A useful estimate for the value of a slope at 0, plus some type information for slopes.

```
lemma (in int1) Int_ZF_2_1_L8: assumes A1: s∈𝒮
  shows
  abs(s(0)) ≤ maxδ(s)
  0 ≤ maxδ(s)
  abs(s(0)) ∈ ℤ   maxδ(s) ∈ ℤ
  abs(s(0)) + maxδ(s) ∈ ℤ
proof -
  from A1 have s(0) ∈ ℤ
    using int_zero_one_are_int Int_ZF_2_1_L2B by simp
  then have I: 0 ≤ abs(s(0))
    and abs(δ(s,0,0)) = abs(s(0))
    using int_abs_nonneg int_zero_one_are_int Int_ZF_1_1_L4
      Int_ZF_2_L17 by auto
  moreover from A1 have abs(δ(s,0,0)) ≤ maxδ(s)
    using int_zero_one_are_int Int_ZF_2_1_L7 by simp
  ultimately show II: abs(s(0)) ≤ maxδ(s)
    by simp
  with I show 0≤maxδ(s) by (rule Int_order_transitive)
  with II show
    maxδ(s) ∈ ℤ   abs(s(0)) ∈ ℤ
    abs(s(0)) + maxδ(s) ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
qed
```

Int `Group_ZF_3.thy` we show that finite range functions valued in an abelian group form a normal subgroup of almost homomorphisms. This allows to define the equivalence relation between almost homomorphisms as the relation resulting from dividing by that normal subgroup. Then we show in `Group_ZF_3_4_L12` that if the difference of f and g has finite range (actually $f(n) \cdot g(n)^{-1}$ as we use multiplicative notation in `Group_ZF_3.thy`), then f and g are equivalent. The next lemma translates that fact into the notation used in `int1` context.

```
lemma (in int1) Int_ZF_2_1_L9: assumes A1: s∈𝒮  r∈𝒮
  and A2: ∀m∈ℤ. abs(s(m)-r(m)) ≤ L
  shows s ∼ r
proof -
  from A1 A2 have
    ∀m∈ℤ. s(m)-r(m) ∈ ℤ ∧ abs(s(m)-r(m)) ≤ L
    using Int_ZF_2_1_L2B Int_ZF_1_1_L5 by simp
  then have
    IsBounded({s(n)-r(n). n∈ℤ}, IntegerOrder)
```

```
    by (rule Int_ZF_1_3_L20)
  with A1 show s ∼ r using Int_bounded_iff_fin
    Int_ZF_2_1_L1 group1.Group_ZF_3_4_L12 by simp
qed
```

A neccessary condition for two slopes to be almost equal. For slopes the definition postulates the set $\{f(m) - g(m) : m \in Z\}$ to be finite. This lemma shows that this imples that $|f(m) - g(m)|$ is bounded (by some integer) as m varies over integers. We also mention here that in this context `s ∼ r` implies that both s and r are slopes.

```
lemma (in int1) Int_ZF_2_1_L9A: assumes s ∼ r
  shows
  ∃L∈ℤ. ∀m∈ℤ. abs(s(m)-r(m)) ≤ L
  s∈𝒮  r∈𝒮
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_4_L11
    Int_ZF_1_3_L20AA QuotientGroupRel_def by auto
```

Let's recall that the relation of almost equality is an equivalence relation on the set of slopes.

```
lemma (in int1) Int_ZF_2_1_L9B: shows
  AlEqRel ⊆ 𝒮×𝒮
  equiv(𝒮,AlEqRel)
  using Int_ZF_2_1_L1 group1.Group_ZF_3_3_L3 by auto
```

Another version of sufficient condition for two slopes to be almost equal: if the difference of two slopes is a finite range function, then they are almost equal.

```
lemma (in int1) Int_ZF_2_1_L9C: assumes s∈𝒮  r∈𝒮 and
  s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
  shows
  s ∼ r
  r ∼ s
  using prems Int_ZF_2_1_L1
    group1.Group_ZF_3_2_L13 group1.Group_ZF_3_4_L12A
  by auto
```

If two slopes are almost equal, then the difference has finite range. This is the inverse of `Int_ZF_2_1_L9C`.

```
lemma (in int1) Int_ZF_2_1_L9D: assumes A1: s ∼ r
  shows s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  let G = ℤ
  let f = IntegerAddition
  from A1 have AlHomOp1(G, f)⟨s,GroupInv(AlmostHoms(G, f),AlHomOp1(G,
f))(r)⟩
    ∈ FinRangeFunctions(G, G)
    using Int_ZF_2_1_L1 group1.Group_ZF_3_4_L12B by auto
```

```
  with A1 show s + (-r) ∈ FinRangeFunctions(ℤ,ℤ)
    using Int_ZF_2_1_L9A Int_ZF_2_1_L1 group1.Group_ZF_3_2_L13
    by simp
qed
```

What is the value of a composition of slopes?

```
lemma (in int1) Int_ZF_2_1_L10:
  assumes s∈𝒮  r∈𝒮 and m∈ℤ
  shows (s∘r)(m) = s(r(m))  s(r(m)) ∈ ℤ
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_4_L2 by auto
```

Composition of slopes is a slope.

```
lemma (in int1) Int_ZF_2_1_L11:
  assumes s∈𝒮  r∈𝒮
  shows s∘r ∈ 𝒮
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_4_T1 by simp
```

Negative of a slope is a slope.

```
lemma (in int1) Int_ZF_2_1_L12: assumes s∈𝒮 shows -s ∈ 𝒮
  using prems Int_ZF_1_T2 Int_ZF_2_1_L1 group1.Group_ZF_3_2_L13
  by simp
```

What is the value of a negative of a slope?

```
lemma (in int1) Int_ZF_2_1_L12A:
  assumes s∈𝒮 and m∈ℤ shows (-s)(m) = -(s(m))
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_2_L5
  by simp
```

What are the values of a sum of slopes?

```
lemma (in int1) Int_ZF_2_1_L12B: assumes s∈𝒮  r∈𝒮 and m∈ℤ
  shows (s+r)(m) = s(m) + r(m)
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_2_L12
  by simp
```

Sum of slopes is a slope.

```
lemma (in int1) Int_ZF_2_1_L12C: assumes s∈𝒮  r∈𝒮
  shows s+r ∈ 𝒮
  using prems Int_ZF_2_1_L1 group1.Group_ZF_3_2_L16
  by simp
```

A simple but useful identity.

```
lemma (in int1) Int_ZF_2_1_L13:
  assumes s∈𝒮 and n∈ℤ  m∈ℤ
  shows s(n·m) + (s(m) + δ(s,n·m,m)) = s((n+1)·m)
  using prems Int_ZF_1_1_L5 Int_ZF_2_1_L2B Int_ZF_1_2_L9 Int_ZF_1_2_L7
  by simp
```

Some estimates for the absolute value of a slope at the opposite integer.

```
lemma (in int1) Int_ZF_2_1_L14: assumes A1: s∈𝒮 and A2: m∈ℤ
  shows
  s(-m) = s(0) - δ(s,m,-m) - s(m)
  abs(s(m)+s(-m)) ≤ 2·maxδ(s)
  abs(s(-m)) ≤ 2·maxδ(s) + abs(s(m))
  s(-m) ≤ abs(s(0)) + maxδ(s) - s(m)
proof -
  from A1 A2 have T:
    (-m) ∈ ℤ  abs(s(m)) ∈ ℤ  s(0) ∈ ℤ  abs(s(0)) ∈ ℤ
    δ(s,m,-m) ∈ ℤ   s(m) ∈ ℤ   s(-m) ∈ ℤ
    (-(s(m))) ∈ ℤ  s(0) - δ(s,m,-m) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_2_1_L2B Int_ZF_2_L14 Int_ZF_2_1_L2
      Int_ZF_1_1_L5 int_zero_one_are_int by auto
  with A2 show I: s(-m) = s(0) - δ(s,m,-m) - s(m)
    using Int_ZF_1_1_L4 Int_ZF_1_2_L15 by simp
  from T have abs(s(0) - δ(s,m,-m)) ≤ abs(s(0)) + abs(δ(s,m,-m))
    using Int_triangle_ineq1 by simp
  moreover from A1 A2 T have abs(s(0)) + abs(δ(s,m,-m)) ≤  2·maxδ(s)
    using Int_ZF_2_1_L7 Int_ZF_2_1_L8 Int_ZF_1_3_L21 by simp
  ultimately have abs(s(0) - δ(s,m,-m)) ≤ 2·maxδ(s)
    by (rule Int_order_transitive)
  moreover
  from I have s(m) + s(-m) = s(m) + (s(0) - δ(s,m,-m) - s(m))
    by simp
  with T have abs(s(m) + s(-m)) = abs(s(0) - δ(s,m,-m))
    using Int_ZF_1_2_L3 by simp
  ultimately show abs(s(m)+s(-m)) ≤ 2·maxδ(s)
    by simp
  from I have abs(s(-m)) = abs(s(0) - δ(s,m,-m) - s(m))
    by simp
  with T have
    abs(s(-m)) ≤ abs(s(0)) + abs(δ(s,m,-m)) + abs(s(m))
    using int_triangle_ineq3 by simp
  moreover from A1 A2 T have
    abs(s(0)) + abs(δ(s,m,-m)) + abs(s(m)) ≤ 2·maxδ(s) + abs(s(m))
    using Int_ZF_2_1_L7 Int_ZF_2_1_L8 Int_ZF_1_3_L21 int_ord_transl_inv
by simp
  ultimately show abs(s(-m)) ≤ 2·maxδ(s) + abs(s(m))
    by (rule Int_order_transitive)
  from T have s(0) - δ(s,m,-m) ≤ abs(s(0)) + abs(δ(s,m,-m))
    using Int_ZF_2_L15E by simp
  moreover from A1 A2 T have
    abs(s(0)) + abs(δ(s,m,-m)) ≤ abs(s(0)) + maxδ(s)
    using Int_ZF_2_1_L7 int_ord_transl_inv by simp
  ultimately have s(0) - δ(s,m,-m) ≤ abs(s(0)) + maxδ(s)
    by (rule Int_order_transitive)
  with T have
    s(0) - δ(s,m,-m) - s(m) ≤ abs(s(0)) + maxδ(s) - s(m)
    using int_ord_transl_inv by simp
```

```
  with I show s(-m) ≤ abs(s(0)) + maxδ(s) - s(m)
    by simp
qed
```

An identity that expresses the value of an integer function at the opposite integer in terms of the value of that function at the integer, zero, and the homomorphism difference. We have a similar identity in `Int_ZF_2_1_L14`, but over there we assume that f is a slope.

```
lemma (in int1) Int_ZF_2_1_L14A: assumes A1: f:ℤ→ℤ and A2: m∈ℤ
  shows f(-m) = (-δ(f,m,-m)) + f(0) - f(m)
proof -
  from A1 A2 have T:
    f(-m) ∈ ℤ  δ(f,m,-m) ∈ ℤ  f(0) ∈ ℤ  f(m) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_1_1_L5 int_zero_one_are_int apply_funtype

    by auto
   with A2 show f(-m) = (-δ(f,m,-m)) + f(0) - f(m)
     using Int_ZF_1_1_L4 Int_ZF_1_2_L15 by simp
qed
```

The next lemma allows to use the expression `maxf(f,0..M-1)`. Recall that `maxf(f,A)` is the maximum of (function) f on (the set) A.

```
lemma (in int1) Int_ZF_2_1_L15:
  assumes s∈𝒮 and M ∈ ℤ₊
  shows
  maxf(s,0..(M-1)) ∈ ℤ
  ∀n ∈ 0..(M-1). s(n) ≤ maxf(s,0..(M-1))
  minf(s,0..(M-1)) ∈ ℤ
  ∀n ∈ 0..(M-1). minf(s,0..(M-1)) ≤ s(n)
  using prems AlmostHoms_def Int_ZF_1_5_L6 Int_ZF_1_4_L2
  by auto
```

A lower estimate for the value of a slope at $nM + k$.

```
lemma (in int1) Int_ZF_2_1_L16:
  assumes A1: s∈𝒮  and A2: m∈ℤ and A3: M ∈ ℤ₊ and A4: k ∈ 0..(M-1)
  shows s(m·M) + (minf(s,0..(M-1))- maxδ(s)) ≤ s(m·M+k)
proof -
  from A3 have 0..(M-1) ⊆ ℤ
    using Int_ZF_1_5_L6 by simp
  with A1 A2 A3 A4 have T: m·M ∈ ℤ   k ∈ ℤ  s(m·M) ∈ ℤ
    using PositiveSet_def Int_ZF_1_1_L5  Int_ZF_2_1_L2B
    by auto
  with A1 A3 A4 have
    s(m·M) + (minf(s,0..(M-1)) - maxδ(s)) ≤ s(m·M) + (s(k) + δ(s,m·M,k))
    using Int_ZF_2_1_L15 Int_ZF_2_1_L7 int_ineq_add_sides int_ord_transl_inv
    by simp
  with A1 T show thesis using Int_ZF_2_1_L3A by simp
qed
```

Identity is a slope.

```
lemma (in int1) Int_ZF_2_1_L17: shows id(ℤ) ∈ 𝒮
  using Int_ZF_2_1_L1 group1.Group_ZF_3_4_L15 by simp
```

Simple identities about (absolute value of) homomorphism differences.

```
lemma (in int1) Int_ZF_2_1_L18:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  abs(f(n) + f(m) - f(m+n)) = abs(δ(f,m,n))
  abs(f(m) + f(n) - f(m+n)) = abs(δ(f,m,n))
  (-(f(m))) - f(n) + f(m+n) = δ(f,m,n)
  (-(f(n))) - f(m) + f(m+n) = δ(f,m,n)
  abs((-f(m+n)) + f(m) + f(n)) = abs(δ(f,m,n))
proof -
  from A1 A2 have T:
    f(m+n) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ
    f(m+n) - f(m) -  f(n)  ∈ ℤ
    (-(f(m))) ∈ ℤ
    (-f(m+n)) + f(m) + f(n) ∈ ℤ
    using apply_funtype Int_ZF_1_1_L4 Int_ZF_1_1_L5 by auto
  then have
    abs(-(f(m+n) - f(m) -  f(n))) = abs(f(m+n) - f(m) -  f(n))
    using Int_ZF_2_L17 by simp
  moreover from T have
    (-(f(m+n) - f(m) -  f(n))) = f(n) + f(m) - f(m+n)
    using Int_ZF_1_2_L9A by simp
  ultimately show abs(f(n) + f(m) - f(m+n)) = abs(δ(f,m,n))
    by simp
  moreover from T have f(n) + f(m) = f(m) + f(n)
    using Int_ZF_1_1_L5 by simp
  ultimately show abs(f(m) + f(n) - f(m+n)) = abs(δ(f,m,n))
    by simp
  from T show
    (-(f(m))) - f(n) + f(m+n) = δ(f,m,n)
    (-(f(n))) - f(m) + f(m+n) = δ(f,m,n)
    using Int_ZF_1_2_L9 by auto
  from T have
    abs((-f(m+n)) + f(m) + f(n)) =
    abs(-((-f(m+n)) + f(m) + f(n)))
    using Int_ZF_2_L17 by simp
  also from T have
    abs(-((-f(m+n)) + f(m) + f(n))) = abs(δ(f,m,n))
    using Int_ZF_1_2_L9 by simp
  finally show abs((-f(m+n)) + f(m) + f(n)) = abs(δ(f,m,n))
    by simp
qed
```

Some identities about the homomorphism difference of odd functions.

```
lemma (in int1) Int_ZF_2_1_L19:
```

```
  assumes A1: f:ℤ→ℤ and A2: ∀x∈ℤ. (-f(-x)) = f(x)
  and A3: m∈ℤ  n∈ℤ
  shows
  abs(δ(f,-m,m+n)) = abs(δ(f,m,n))
  abs(δ(f,-n,m+n)) = abs(δ(f,m,n))
  δ(f,n,-(m+n)) = δ(f,m,n)
  δ(f,m,-(m+n)) = δ(f,m,n)
  abs(δ(f,-m,-n)) = abs(δ(f,m,n))
proof -
  from A1 A2 A3 show
    abs(δ(f,-m,m+n)) = abs(δ(f,m,n))
    abs(δ(f,-n,m+n)) = abs(δ(f,m,n))
    using Int_ZF_1_2_L3 Int_ZF_2_1_L18 by auto
  from A3 have T: m+n ∈ ℤ using Int_ZF_1_1_L5 by simp
  from A1 A2 have I: ∀x∈ℤ. f(-x) = (-f(x))
    using Int_ZF_1_5_L13 by simp
  with A1 A2 A3 T show
    δ(f,n,-(m+n)) = δ(f,m,n)
    δ(f,m,-(m+n)) = δ(f,m,n)
    using Int_ZF_1_2_L3 Int_ZF_2_1_L18 by auto
  from A3 have
    abs(δ(f,-m,-n)) = abs(f(-(m+n)) - f(-m) - f(-n))
    using Int_ZF_1_1_L5 by simp
  also from A1 A2 A3 T I have ... = abs(δ(f,m,n))
    using Int_ZF_2_1_L18 by simp
  finally show abs(δ(f,-m,-n)) = abs(δ(f,m,n)) by simp
qed
```

Recall that f is a slope iff $f(m+n)-f(m)-f(n)$ is bounded as m,n ranges over integers. The next lemma is the first step in showing that we only need to check this condition as m,n ranges over positive intergers. Namely we show that if the condition holds for positive integers, then it holds if one integer is positive and the second one is nonnegative.

```
lemma (in int1) Int_ZF_2_1_L20: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L and
  A3:  m∈ℤ⁺  n∈ℤ₊
  shows
  0 ≤ L
  abs(δ(f,m,n)) ≤ L + abs(f(0))
proof -
  from A1 A2 have
    δ(f,1,1) ∈ ℤ  and abs(δ(f,1,1)) ≤ L
    using int_one_two_are_pos PositiveSet_def Int_ZF_2_1_L3B
    by auto
  then show I: 0 ≤ L using Int_ZF_1_3_L19 by simp
  from A1 A3 have T:
    n ∈ ℤ  f(n) ∈ ℤ  f(0) ∈ ℤ
    δ(f,m,n) ∈ ℤ  abs(δ(f,m,n)) ∈ ℤ
    using PositiveSet_def int_zero_one_are_int apply_funtype
```

```
    Nonnegative_def Int_ZF_2_1_L3B Int_ZF_2_L14 by auto
  from A3 have m=0 ∨ m∈ℤ₊ using Int_ZF_1_5_L3A by auto
  moreover
  { assume m = 0
    with T I have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using Int_ZF_1_1_L4 Int_ZF_1_2_L3 Int_ZF_2_L17
        int_ord_is_refl refl_def Int_ZF_2_L15F by simp }
  moreover
  { assume m∈ℤ₊
    with A2 A3 T have abs(δ(f,m,n)) ≤ L + abs(f(0))
       using int_abs_nonneg Int_ZF_2_L15F by simp }
   ultimately show abs(δ(f,m,n)) ≤ L + abs(f(0))
     by auto
qed
```

If the slope condition holds for all pairs of integers such that one integer is positive and the second one is nonnegative, then it holds when both integers are nonnegative.

```
lemma (in int1) Int_ZF_2_1_L21: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ⁺. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L and
  A3: n∈ℤ⁺  m∈ℤ⁺
  shows abs(δ(f,m,n)) ≤ L + abs(f(0))
proof -
  from A1 A2 have
    δ(f,1,1) ∈ ℤ  and abs(δ(f,1,1)) ≤ L
    using int_one_two_are_pos PositiveSet_def Nonnegative_def Int_ZF_2_1_L3B
    by auto
  then have I: 0 ≤ L using Int_ZF_1_3_L19 by simp
  from A1 A3 have T:
    m ∈ ℤ  f(m) ∈ ℤ  f(0) ∈ ℤ  (-f(0)) ∈ ℤ
    δ(f,m,n) ∈ ℤ  abs(δ(f,m,n)) ∈ ℤ
    using int_zero_one_are_int apply_funtype Nonnegative_def
      Int_ZF_2_1_L3B Int_ZF_2_L14 Int_ZF_1_1_L4 by auto
  from A3 have n=0 ∨ n∈ℤ₊ using Int_ZF_1_5_L3A by auto
  moreover
  { assume n=0
     with T have δ(f,m,n) = -f(0)
      using Int_ZF_1_1_L4 by simp
    with T have abs(δ(f,m,n)) = abs(f(0))
      using Int_ZF_2_L17 by simp
    with T have abs(δ(f,m,n)) ≤ abs(f(0))
      using int_ord_is_refl refl_def by simp
    with T I have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using Int_ZF_2_L15F by simp }
  moreover
  { assume n∈ℤ₊
    with A2 A3 T have abs(δ(f,m,n)) ≤ L + abs(f(0))
      using int_abs_nonneg Int_ZF_2_L15F by simp }
  ultimately show  abs(δ(f,m,n)) ≤ L + abs(f(0))
```

```
    by auto
qed
```

If the homomorphism difference is bounded on $\mathbb{Z}_+ \times \mathbb{Z}_+$, then it is bounded on $\mathbb{Z}^+ \times \mathbb{Z}^+$.

```
lemma (in int1) Int_ZF_2_1_L22: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  shows ∃M. ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ M
proof -
  from A1 A2 have
    ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ L + abs(f(0)) + abs(f(0))
    using Int_ZF_2_1_L20 Int_ZF_2_1_L21 by simp
  then show thesis by auto
qed
```

For odd functions we can do better than in Int_ZF_2_1_L22: if the homomorphism difference of f is bounded on $\mathbb{Z}^+ \times \mathbb{Z}^+$, then it is bounded on $\mathbb{Z} \times \mathbb{Z}$, hence f is a slope. Loong prof by splitting the $\mathbb{Z} \times \mathbb{Z}$ into six subsets.

```
lemma (in int1) Int_ZF_2_1_L23: assumes A1: f:ℤ→ℤ and
  A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  and A3: ∀x∈ℤ. (-f(-x)) = f(x)
  shows  f∈𝒮
proof -
  from A1 A2 have
    ∃M.∀a∈ℤ⁺. ∀b∈ℤ⁺. abs(δ(f,a,b)) ≤ M
    by (rule Int_ZF_2_1_L22)
  then obtain M where I: ∀m∈ℤ⁺. ∀n∈ℤ⁺. abs(δ(f,m,n)) ≤ M
    by auto
  { fix a b assume A4: a∈ℤ  b∈ℤ
    then have
      0≤a ∧ 0≤b  ∨  a≤0 ∧ b≤0  ∨
      a≤0 ∧ 0≤b ∧ 0 ≤ a+b  ∨ a≤0 ∧ 0≤b ∧ a+b ≤ 0  ∨
      0≤a ∧ b≤0 ∧ 0 ≤ a+b  ∨  0≤a ∧ b≤0 ∧ a+b ≤ 0
      using int_plane_split_in6 by simp
    moreover
    { assume 0≤a ∧ 0≤b
      then have a∈ℤ⁺  b∈ℤ⁺
        using Int_ZF_2_L16 by auto
      with I have abs(δ(f,a,b)) ≤ M by simp }
    moreover
    { assume a≤0 ∧ b≤0
      with I have abs(δ(f,-a,-b)) ≤ M
        using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
        using Int_ZF_2_1_L19 by simp }
    moreover
    { assume a≤0 ∧ 0≤b ∧ 0 ≤ a+b
      with I have abs(δ(f,-a,a+b)) ≤ M
        using Int_ZF_2_L10A Int_ZF_2_L16 by simp
```

```
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
        using Int_ZF_2_1_L19 by simp }
    moreover
    { assume a≤0 ∧ 0≤b ∧ a+b ≤ 0
      with I have abs(δ(f,b,-(a+b))) ≤ M
        using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
        using Int_ZF_2_1_L19 by simp }
    moreover
    { assume 0≤a ∧ b≤0 ∧ 0 ≤ a+b
      with I have abs(δ(f,-b,a+b)) ≤ M
        using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
        using Int_ZF_2_1_L19 by simp }
    moreover
    { assume 0≤a ∧ b≤0 ∧ a+b ≤ 0
      with I have abs(δ(f,a,-(a+b))) ≤ M
        using Int_ZF_2_L10A Int_ZF_2_L16 by simp
      with A1 A3 A4 have abs(δ(f,a,b)) ≤ M
        using Int_ZF_2_1_L19 by simp }
    ultimately have abs(δ(f,a,b)) ≤ M by auto }
  then have ∀m∈ℤ. ∀n∈ℤ. abs(δ(f,m,n)) ≤ M by simp
  with A1 show f∈𝒮 by (rule Int_ZF_2_1_L5)
qed
```

If the homomorphism difference of a function defined on positive integers is bounded, then the odd extension of this function is a slope.

```
lemma (in int1) Int_ZF_2_1_L24:
  assumes A1: f:ℤ₊→ℤ and A2: ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,f) ∈ 𝒮
proof -
  let g = OddExtension(ℤ,IntegerAddition,IntegerOrder,f)
  from A1 have g : ℤ→ℤ
    using Int_ZF_1_5_L10 by simp
  moreover have ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(g,a,b)) ≤ L
  proof -
    { fix a b assume A3: a∈ℤ₊  b∈ℤ₊
      with A1 have abs(δ(f,a,b)) =  abs(δ(g,a,b))
        using pos_int_closed_add_unfolded Int_ZF_1_5_L11
        by simp
      moreover from A2 A3 have abs(δ(f,a,b)) ≤ L by simp
      ultimately have abs(δ(g,a,b)) ≤ L by simp
    } then show thesis by simp
  qed
  moreover from A1 have ∀x∈ℤ. (-g(-x)) = g(x)
    using int_oddext_is_odd_alt by simp
  ultimately show g ∈ 𝒮 by (rule Int_ZF_2_1_L23)
qed
```

Type information related to γ.

```
lemma (in int1) Int_ZF_2_1_L25:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  δ(f,m,-n) ∈ ℤ
  δ(f,n,-n) ∈ ℤ
  (-δ(f,n,-n)) ∈ ℤ
  f(0) ∈ ℤ
  γ(f,m,n)  ∈ ℤ
proof -
  from A1 A2 show T1:
    δ(f,m,-n) ∈ ℤ  f(0) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_2_1_L3B int_zero_one_are_int apply_funtype
    by auto
  from A2 have (-n) ∈ ℤ
    using Int_ZF_1_1_L4 by simp
  with A1 A2 show δ(f,n,-n) ∈ ℤ
    using Int_ZF_2_1_L3B by simp
  then show (-δ(f,n,-n)) ∈ ℤ
    using Int_ZF_1_1_L4 by simp
  with T1 show γ(f,m,n)  ∈ ℤ
    using Int_ZF_1_1_L5 by simp
qed
```

A couple of formulae involving $f(m-n)$ and $\gamma(f,m,n)$.

```
lemma (in int1) Int_ZF_2_1_L26:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ
  shows
  f(m-n) = γ(f,m,n) + f(m) - f(n)
  f(m-n) = γ(f,m,n) + (f(m) - f(n))
  f(m-n) + (f(n) - γ(f,m,n)) = f(m)
proof -
  from A1 A2 have T:
    (-n) ∈ ℤ  δ(f,m,-n) ∈ ℤ
    f(0) ∈ ℤ  f(m) ∈ ℤ  f(n) ∈ ℤ  (-f(n)) ∈ ℤ
    (-δ(f,n,-n)) ∈ ℤ
    (-δ(f,n,-n))  + f(0) ∈ ℤ
    γ(f,m,n) ∈ ℤ
    using  Int_ZF_1_1_L4 Int_ZF_2_1_L25 apply_funtype Int_ZF_1_1_L5
    by auto
   with A1 A2 have f(m-n) =
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0) - f(n)) + f(m)
    using Int_ZF_2_1_L3C Int_ZF_2_1_L14A by simp
  with T have f(m-n) =
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0)) + f(m) - f(n)
    using Int_ZF_1_2_L16 by simp
  moreover from T have
    δ(f,m,-n) + ((-δ(f,n,-n)) + f(0)) = γ(f,m,n)
    using Int_ZF_1_1_L7 by simp
```

```
  ultimately show  I: f(m-n) = γ(f,m,n) + f(m) - f(n)
    by simp
  then have f(m-n) + (f(n) - γ(f,m,n)) =
    (γ(f,m,n) + f(m) - f(n)) + (f(n) - γ(f,m,n))
    by simp
  moreover from T have ... = f(m) using Int_ZF_1_2_L18
    by simp
  ultimately show f(m-n) + (f(n) - γ(f,m,n)) = f(m)
    by simp
  from T have γ(f,m,n) ∈ ℤ  f(m) ∈ ℤ  (-f(n)) ∈ ℤ
    by auto
  then have
    γ(f,m,n) + f(m) + (-f(n)) =  γ(f,m,n) + (f(m) + (-f(n)))
    by (rule Int_ZF_1_1_L7)
  with I show  f(m-n) = γ(f,m,n) + (f(m) - f(n)) by simp
qed
```

A formula expressing the difference between $f(m-n-k)$ and $f(m)-f(n)-f(k)$ in terms of γ.

```
lemma (in int1) Int_ZF_2_1_L26A:
  assumes A1: f:ℤ→ℤ and A2: m∈ℤ  n∈ℤ  k∈ℤ
  shows
  f(m-n-k) - (f(m)- f(n) - f(k)) = γ(f,m-n,k) + γ(f,m,n)
proof -
  from A1 A2 have
    T: m-n ∈ ℤ γ(f,m-n,k) ∈ ℤ  f(m) - f(n) - f(k) ∈ ℤ and
    T1: γ(f,m,n) ∈ ℤ  f(m) - f(n) ∈ ℤ  (-f(k)) ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_1_1_L5 Int_ZF_2_1_L25 apply_funtype
    by auto
  from A1 A2 have
    f(m-n) - f(k) = γ(f,m,n) + (f(m) - f(n)) + (-f(k))
    using Int_ZF_2_1_L26 by simp
  also from T1 have ... = γ(f,m,n) + (f(m) - f(n) + (-f(k)))
    by (rule Int_ZF_1_1_L7)
  finally have
    f(m-n) - f(k) = γ(f,m,n) + (f(m) - f(n) - f(k))
    by simp
  moreover from A1 A2 T have
    f(m-n-k) =  γ(f,m-n,k) + (f(m-n)-f(k))
    using Int_ZF_2_1_L26 by simp
  ultimately have
    f(m-n-k) - (f(m)- f(n) - f(k)) =
    γ(f,m-n,k) + ( γ(f,m,n) + (f(m) - f(n) - f(k)))
    - (f(m)- f(n) - f(k))
    by simp
  with T T1 show thesis
    using Int_ZF_1_2_L17 by simp
qed
```

If s is a slope, then $\gamma(s, m, n)$ is uniformly bounded.

```
lemma (in int1) Int_ZF_2_1_L27: assumes A1: s∈𝒮
  shows ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
proof -
  let L = maxδ(s) + maxδ(s) + abs(s(0))
  from A1 have T:
    maxδ(s) ∈ ℤ  abs(s(0)) ∈ ℤ  L ∈ ℤ
    using Int_ZF_2_1_L8 int_zero_one_are_int Int_ZF_2_1_L2B
      Int_ZF_2_L14 Int_ZF_1_1_L5 by auto
  moreover
  { fix m
    fix n
    assume A2: m∈ℤ  n∈ℤ
    with A1 have T:
      (-n) ∈ ℤ
      δ(s,m,-n) ∈ ℤ
      δ(s,n,-n) ∈ ℤ
      (-δ(s,n,-n)) ∈ ℤ
      s(0) ∈ ℤ  abs(s(0)) ∈ ℤ
      using Int_ZF_1_1_L4 AlmostHoms_def Int_ZF_2_1_L25 Int_ZF_2_L14
      by auto
    with T have
      abs(δ(s,m,-n) - δ(s,n,-n) + s(0)) ≤
      abs(δ(s,m,-n)) + abs(-δ(s,n,-n)) + abs(s(0))
      using Int_triangle_ineq3 by simp
    moreover from A1 A2 T have
      abs(δ(s,m,-n)) + abs(-δ(s,n,-n)) + abs(s(0)) ≤ L
      using Int_ZF_2_1_L7 int_ineq_add_sides int_ord_transl_inv Int_ZF_2_L17
      by simp
    ultimately have abs(δ(s,m,-n) - δ(s,n,-n) + s(0)) ≤ L
      by (rule Int_order_transitive)
    then have abs(γ(s,m,n)) ≤ L by simp }
  ultimately show ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    by auto
qed
```

If s is a slope, then $s(m) \leq s(m-1) + M$, where L does not depend on m.

```
lemma (in int1) Int_ZF_2_1_L28: assumes A1: s∈𝒮
  shows ∃M∈ℤ. ∀m∈ℤ. s(m) ≤ s(m-1) + M
proof -
  from A1 have
    ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ.abs(γ(s,m,n)) ≤ L
    using Int_ZF_2_1_L27 by simp
  then obtain L where T: L∈ℤ and ∀m∈ℤ.∀n∈ℤ.abs(γ(s,m,n)) ≤ L
    using Int_ZF_2_1_L27 by auto
  then have I: ∀m∈ℤ.abs(γ(s,m,1)) ≤ L
    using int_zero_one_are_int by simp
  let M = s(1) + L
  from A1 T have M ∈ ℤ
```

```
    using int_zero_one_are_int Int_ZF_2_1_L2B Int_ZF_1_1_L5
    by simp
  moreover
  { fix m assume A2: m∈ℤ
    with A1 have
      T1: s:ℤ→ℤ  m∈ℤ  1∈ℤ and
      T2: γ(s,m,1) ∈ ℤ  s(1) ∈ ℤ
      using int_zero_one_are_int AlmostHoms_def
        Int_ZF_2_1_L25 by auto
    from A2 T1 have T3: s(m-1) ∈ ℤ
      using Int_ZF_1_1_L5 apply_funtype by simp
    from I A2 T2 have
      (-γ(s,m,1)) ≤ abs(γ(s,m,1))
      abs(γ(s,m,1)) ≤ L
      using Int_ZF_2_L19C by auto
    then have (-γ(s,m,1)) ≤ L
      by (rule Int_order_transitive)
    with T2 T3 have
      s(m-1) + (s(1) - γ(s,m,1)) ≤ s(m-1) + M
      using int_ord_transl_inv by simp
    moreover from T1 have
      s(m-1) + (s(1) - γ(s,m,1)) = s(m)
      by (rule Int_ZF_2_1_L26)
    ultimately have s(m) ≤ s(m-1) + M  by simp  }
  ultimately show ∃M∈ℤ. ∀m∈ℤ. s(m) ≤ s(m-1) + M
    by auto
qed
```

If s is a slope, then the difference between $s(m-n-k)$ and $s(m)-s(n)-s(k)$ is uniformly bounded.

```
lemma (in int1) Int_ZF_2_1_L29: assumes A1: s∈𝒮
  shows
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
proof -
  from A1 have ∃L∈ℤ. ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    using Int_ZF_2_1_L27 by simp
  then obtain L where I: L∈ℤ and
    II: ∀m∈ℤ.∀n∈ℤ. abs(γ(s,m,n)) ≤ L
    by auto
  from I have L+L ∈ ℤ
    using Int_ZF_1_1_L5 by simp
  moreover
  { fix m n k assume A2: m∈ℤ  n∈ℤ  k∈ℤ
    with A1 have T:
      m-n ∈ ℤ  γ(s,m-n,k) ∈ ℤ  γ(s,m,n) ∈ ℤ
      using Int_ZF_1_1_L5 AlmostHoms_def Int_ZF_2_1_L25
      by auto
    then have
      I: abs(γ(s,m-n,k) + γ(s,m,n)) ≤ abs(γ(s,m-n,k)) + abs(γ(s,m,n))
```

```
      using Int_triangle_ineq by simp
    from II A2 T have
      abs(γ(s,m-n,k)) ≤ L
      abs(γ(s,m,n)) ≤ L
      by auto
    then have abs(γ(s,m-n,k)) + abs(γ(s,m,n)) ≤ L+L
      using int_ineq_add_sides by simp
    with I have abs(γ(s,m-n,k) + γ(s,m,n)) ≤ L+L
      by (rule Int_order_transitive)
    moreover from A1 A2 have
      s(m-n-k) - (s(m)- s(n) - s(k)) = γ(s,m-n,k) + γ(s,m,n)
      using AlmostHoms_def Int_ZF_2_1_L26A by simp
    ultimately have
      abs(s(m-n-k) - (s(m)- s(n) - s(k))) ≤ L+L
      by simp }
  ultimately show thesis by auto
qed
```

If s is a slope, then we can find integers M, K such that $s(m-n-k) \leq s(m)-s(n)-s(k)+M$ and $s(m)-s(n)-s(k)+K \leq s(m-n-k)$, for all integer m, n, k.

```
lemma (in int1) Int_ZF_2_1_L30: assumes A1: s∈𝒮
  shows
  ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
  ∃K∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k)+K ≤ s(m-n-k)
proof -
  from A1 have
    ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    using Int_ZF_2_1_L29 by simp
  then obtain M where I: M∈ℤ and II:
    ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
    by auto
  from I have III: (-M) ∈ ℤ using Int_ZF_1_1_L4 by simp
  { fix m n k assume A2: m∈ℤ  n∈ℤ  k∈ℤ
    with A1 have s(m-n-k) ∈ ℤ  and s(m)-s(n)-s(k) ∈ ℤ
      using Int_ZF_1_1_L5 Int_ZF_2_1_L2B by auto
    moreover from II A2 have
      abs(s(m-n-k) - (s(m)-s(n)-s(k))) ≤M
      by simp
    ultimately have
      s(m-n-k) ≤ s(m)-s(n)-s(k)+M ∧
      s(m)-s(n)-s(k) - M ≤ s(m-n-k)
      using Int_triangle_ineq2 by simp
  } then have
      ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
      ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k) - M ≤ s(m-n-k)
    by auto
  with I III show
    ∃M∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m-n-k) ≤ s(m)-s(n)-s(k)+M
```

```
    ∃K∈ℤ. ∀m∈ℤ.∀n∈ℤ.∀k∈ℤ. s(m)-s(n)-s(k)+K ≤ s(m-n-k)
    by auto
qed
```

By definition functions f, g are almost equal if $f - g^*$ is bounded. In the next lemma we show it is sufficient to check the boundedness on positive integers.

```
lemma (in int1) Int_ZF_2_1_L31: assumes A1: s∈𝒮  r∈𝒮
  and A2: ∀m∈ℤ₊. abs(s(m)-r(m)) ≤ L
  shows s ∼ r
proof -
  let a = abs(s(0) - r(0))
  let c = 2·maxδ(s) + 2·maxδ(r) + L
  let M = Maximum(IntegerOrder,{a,L,c})
  from A2 have abs(s(1)-r(1)) ≤ L
    using int_one_two_are_pos by simp
  then have T: L∈ℤ using Int_ZF_2_L1A by simp
  moreover from A1 have a ∈ ℤ
    using int_zero_one_are_int Int_ZF_2_1_L2B
      Int_ZF_1_1_L5 Int_ZF_2_L14 by simp
  moreover from A1 T have c ∈ ℤ
    using Int_ZF_2_1_L8 int_two_three_are_int Int_ZF_1_1_L5
    by simp
  ultimately have
    I: a ≤ M and
    II: L ≤ M and
    III: c ≤ M
    using Int_ZF_1_4_L1A by auto

  { fix m assume A5: m∈ℤ
    with A1 have T:
      s(m) ∈ ℤ  r(m) ∈ ℤ  s(m) - r(m) ∈ ℤ
      s(-m) ∈ ℤ  r(-m) ∈ ℤ
      using Int_ZF_2_1_L2B Int_ZF_1_1_L4 Int_ZF_1_1_L5
      by auto
    from A5 have m=0 ∨ m∈ℤ₊ ∨ (-m) ∈ ℤ₊
      using int_decomp_cases by simp
    moreover
    { assume m=0
      with I have abs(s(m) - r(m)) ≤ M
        by simp }
    moreover
    { assume m∈ℤ₊
      with A2 II have
        abs(s(m)-r(m)) ≤ L and L≤M
        by auto
      then have abs(s(m)-r(m)) ≤ M
        by (rule Int_order_transitive) }
    moreover
```

```
    { assume A6: (-m) ∈ ℤ+
      from T have abs(s(m)-r(m)) ≤
        abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m))
        using Int_ZF_1_3_L22A by simp
      moreover
      from A1 A2 III A5 A6 have
        abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m)) ≤ c
        c ≤ M
        using Int_ZF_2_1_L14 int_ineq_add_sides by auto
      then have
        abs(s(m)+s(-m)) + abs(r(m)+r(-m)) + abs(s(-m)-r(-m)) ≤ M
        by (rule Int_order_transitive)
      ultimately have  abs(s(m)-r(m)) ≤ M
        by (rule Int_order_transitive) }
    ultimately have abs(s(m) - r(m)) ≤ M
      by auto
  } then have ∀m∈ℤ. abs(s(m)-r(m)) ≤ M
    by simp
  with A1 show s ∼ r by (rule Int_ZF_2_1_L9)
qed
```

A sufficient condition for an odd slope to be almost equal to identity: If for all positive integers the value of the slope at m is between m and m plus some constant independent of m, then the slope is almost identity.

```
lemma (in int1) Int_ZF_2_1_L32: assumes A1: s∈𝒮  M∈ℤ
  and A2: ∀m∈ℤ+. m ≤ s(m) ∧ s(m) ≤ m+M
  shows s ∼ id(ℤ)
proof -
  let r = id(ℤ)
  from A1 have s∈𝒮  r ∈ 𝒮
    using Int_ZF_2_1_L17 by auto
  moreover from A1 A2 have ∀m∈ℤ+. abs(s(m)-r(m)) ≤ M
    using Int_ZF_1_3_L23 PositiveSet_def id_conv by simp
  ultimately show s ∼ id(ℤ) by (rule Int_ZF_2_1_L31)
qed
```

A lemma about adding a constant to slopes. This is actually proven in `Group_ZF_3_5_L1`, in `Group_ZF_3.thy` here we just refer to that lemma to show it in notation used for integers. Unfortunately we have to use raw set notation in the proof.

```
lemma (in int1) Int_ZF_2_1_L33:
  assumes A1: s∈𝒮 and A2: c∈ℤ and
  A3: r = {⟨m,s(m)+c⟩. m∈ℤ}
  shows
  ∀m∈ℤ. r(m) = s(m)+c
  r∈𝒮
  s ∼ r
proof -
```

```
  let G = ℤ
  let f = IntegerAddition
  let AH =  AlmostHoms(G, f)
  from prems have I:
    group1(G, f)
    s ∈ AlmostHoms(G, f)
    c ∈ G
    r = {⟨x, f⟨s(x), c⟩⟩ . x ∈ G}
    using Int_ZF_2_1_L1 by auto
  then have ∀x∈G. r(x) = f⟨s(x),c⟩
    by (rule group1.Group_ZF_3_5_L1)
  moreover from I have r ∈ AlmostHoms(G, f)
    by (rule group1.Group_ZF_3_5_L1)
  moreover from I have
    ⟨s, r⟩ ∈ QuotientGroupRel(AlmostHoms(G, f), AlHomOp1(G, f), FinRangeFunctions(G,
G))
    by (rule group1.Group_ZF_3_5_L1)
  ultimately show
    ∀m∈ℤ. r(m) = s(m)+c
    r∈𝒮
    s ∼ r
    by auto
qed
```

26.2 Composing slopes

Composition of slopes is not commutative. However, as we show in this section if f and g are slopes then the range of $f \circ g - g \circ f$ is bounded. This allows to show that the multiplication of real numbers is commutative.

Two useful estimates.

```
lemma (in int1) Int_ZF_2_2_L1:
  assumes A1: f:ℤ→ℤ and A2: p∈ℤ  q∈ℤ
  shows
  abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
  abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
proof -
  let R = ℤ
  let A = IntegerAddition
  let M = IntegerMultiplication
  let I = GroupInv(R, A)
  let a = f((p+1)·q)
  let b = p
  let c = f(q)
  let d = f(p·q)
  from A1 A2 have T1:
    ring0(R, A, M)  a ∈ R  b ∈ R  c ∈ R  d ∈ R
    using  Int_ZF_1_1_L2 int_zero_one_are_int Int_ZF_1_1_L5 apply_funtype
```

```
    by auto
  then have
    A⟨a,I(M⟨A⟨b, TheNeutralElement(R, M)⟩,c⟩)⟩ =
    A⟨A⟨A⟨a,I(d)⟩,I(c)⟩,A⟨d, I(M⟨b, c⟩)⟩⟩
    by (rule ring0.Ring_ZF_2_L2)
  with A2 have
    f((p+1)·q)-(p+1)·f(q) = δ(f,p·q,q)+(f(p·q)-p·f(q))
    using int_zero_one_are_int Int_ZF_1_1_L1 Int_ZF_1_1_L4 by simp
  moreover from A1 A2 T1 have δ(f,p·q,q) ∈ ℤ f(p·q)-p·f(q) ∈ ℤ
    using Int_ZF_1_1_L5 apply_funtype by auto
  ultimately show
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
    using Int_triangle_ineq by simp
  from A1 A2 have T1:
    f((p-1)·q) ∈ ℤ   p∈ℤ  f(q) ∈ ℤ   f(p·q) ∈ ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 apply_funtype by auto
  then have
    f((p-1)·q)-(p-1)·f(q) = (f(p·q)-p·f(q))-(f(p·q)-f((p-1)·q)-f(q))
    by (rule Int_ZF_1_2_L6)
  with A2 have f((p-1)·q)-(p-1)·f(q) = (f(p·q)-p·f(q))-δ(f,(p-1)·q,q)
    using Int_ZF_1_2_L7 by simp
  moreover from A1 A2 have
    f(p·q)-p·f(q) ∈ ℤ   δ(f,(p-1)·q,q) ∈ ℤ
    using Int_ZF_1_1_L5 int_zero_one_are_int apply_funtype by auto
  ultimately show
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
    using Int_triangle_ineq1 by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1)\cdot\mathtt{max}\delta(\mathtt{f})$. 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 -
  from A2 have q∈ℤ  p·q ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  with A1 have I: abs(δ(f,p·q,q)) ≤ maxδ(f) by (rule Int_ZF_2_1_L7)
  moreover from A3 have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) .
  moreover from A1 A2 have
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ abs(δ(f,p·q,q))+abs(f(p·q)-p·f(q))
    using AlmostHoms_def Int_ZF_2_L1A Int_ZF_2_2_L1 by simp
  ultimately have
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ maxδ(f)+(abs(p)+1)·maxδ(f)
    by (rule Int_ZF_2_L15)
  moreover from I A2 have
```

```
    maxδ(f)+(abs(p)+1)·maxδ(f) = (abs(p+1)+ 1)·maxδ(f)
    using Int_ZF_2_L1A Int_ZF_1_2_L2 by simp
  ultimately show
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
    by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1)\cdot$maxδ. The proof is by induction on p and the next lemma is the induction step for the case when $p \leq 0$.

```
lemma (in int1) Int_ZF_2_2_L3:
  assumes A1: f∈S and A2: p≤0  q∈ℤ
  and A3: abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
  shows  abs(f((p-1)·q)-(p-1)·f(q)) ≤ (abs(p-1)+ 1)·maxδ(f)
proof -
  from A2 have q∈ℤ  (p-1)·q ∈ ℤ
    using Int_ZF_2_L1A int_zero_one_are_int Int_ZF_1_1_L5 by auto
  with A1 have I: abs(δ(f,(p-1)·q,q)) ≤ maxδ(f) by (rule Int_ZF_2_1_L7)
  moreover from A3 have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) .
  moreover from A1 A2 have
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ abs(δ(f,(p-1)·q,q))+abs(f(p·q)-p·f(q))
    using AlmostHoms_def Int_ZF_2_L1A Int_ZF_2_2_L1 by simp
  ultimately have
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ maxδ(f)+(abs(p)+1)·maxδ(f)
    by (rule Int_ZF_2_L15)
  with I A2 show thesis using Int_ZF_2_L1A Int_ZF_1_2_L5 by simp
qed
```

If f is a slope, then $|f(p \cdot q) - p \cdot f(q)| \leq (|p| + 1)\cdot$maxδ$(f)$.

```
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 (cases 0≤p)
  assume 0≤p
  moreover from A1 A2 have abs(f(0·q)-0·f(q)) ≤ (abs(0)+1)·maxδ(f)
    using int_zero_one_are_int Int_ZF_2_1_L2B Int_ZF_1_1_L4
      Int_ZF_2_1_L8 Int_ZF_2_L18 by simp
  moreover from A1 A2 have
    ∀p. 0≤p ∧ abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) ⟶
    abs(f((p+1)·q)-(p+1)·f(q)) ≤ (abs(p+1)+ 1)·maxδ(f)
    using Int_ZF_2_2_L2 by simp
  ultimately show abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
    by (rule Induction_on_int)
next assume ¬(0≤p)
  with A2 have p≤0 using Int_ZF_2_L19A by simp
  moreover from A1 A2 have abs(f(0·q)-0·f(q)) ≤ (abs(0)+1)·maxδ(f)
    using int_zero_one_are_int Int_ZF_2_1_L2B Int_ZF_1_1_L4
      Int_ZF_2_1_L8 Int_ZF_2_L18 by simp
  moreover from A1 A2 have
```

```
    ∀p. p≤0 ∧ abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f) ⟶
    abs(f((p-1)·q)-(p-1)·f(q)) ≤ (abs(p-1)+ 1)·maxδ(f)
    using Int_ZF_2_2_L3 by simp
  ultimately show abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
    by (rule Back_induct_on_int)
qed
```

The next elegant result is Lemma 7 in the Arthan's paper [2] .

```
lemma (in int1) Arthan_Lem_7:
 assumes A1: f∈𝒮 and A2: p∈ℤ  q∈ℤ
  shows abs(q·f(p)-p·f(q)) ≤ (abs(p)+abs(q)+2)·maxδ(f)
proof -
  from A1 A2 have T:
    q·f(p)-f(p·q) ∈ ℤ
    f(p·q)-p·f(q) ∈ ℤ
    f(q·p) ∈ ℤ  f(p·q) ∈ ℤ
    q·f(p) ∈ ℤ  p·f(q) ∈ ℤ
    maxδ(f) ∈ ℤ
    abs(q) ∈ ℤ  abs(p) ∈ ℤ
    using Int_ZF_1_1_L5 Int_ZF_2_1_L2B Int_ZF_2_1_L7 Int_ZF_2_L14 by auto
  moreover have abs(q·f(p)-f(p·q)) ≤ (abs(q)+1)·maxδ(f)
  proof -
    from A1 A2 have abs(f(q·p)-q·f(p)) ≤ (abs(q)+1)·maxδ(f)
      using Int_ZF_2_2_L4 by simp
    with T A2 show thesis
      using Int_ZF_2_L20 Int_ZF_1_1_L5 by simp
  qed
  moreover from A1 A2 have abs(f(p·q)-p·f(q)) ≤ (abs(p)+1)·maxδ(f)
    using Int_ZF_2_2_L4 by simp
  ultimately have
    abs(q·f(p)-f(p·q)+(f(p·q)-p·f(q))) ≤ (abs(q)+1)·maxδ(f)+(abs(p)+1)·maxδ(f)
    using Int_ZF_2_L21 by simp
  with T show thesis using Int_ZF_1_2_L9 int_zero_one_are_int Int_ZF_1_2_L10
    by simp
qed
```

This is Lemma 8 in the Arthan's paper.

```
lemma (in int1) Arthan_Lem_8: assumes A1: f∈𝒮
  shows ∃A B. A∈ℤ ∧ B∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B)
proof -
  let A = maxδ(f) + abs(f(1))
  let B = 3·maxδ(f)
  from A1 have A∈ℤ B∈ℤ
    using int_zero_one_are_int Int_ZF_1_1_L5 Int_ZF_2_1_L2B
      Int_ZF_2_1_L7 Int_ZF_2_L14 by auto
  moreover have ∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B
  proof
    fix p assume A2: p∈ℤ
    with A1 have T:
```

```
      f(p) ∈ ℤ  abs(p) ∈ ℤ  f(1) ∈ ℤ
      p·f(1) ∈ ℤ  3∈ℤ  maxδ(f) ∈ ℤ
      using Int_ZF_2_1_L2B Int_ZF_2_L14 int_zero_one_are_int
        Int_ZF_1_1_L5 Int_ZF_2_1_L7 by auto
    from A1 A2 have
      abs(1·f(p)-p·f(1)) ≤ (abs(p)+abs(1)+2)·maxδ(f)
      using int_zero_one_are_int Arthan_Lem_7 by simp
    with T have abs(f(p)) ≤ abs(p·f(1))+(abs(p)+3)·maxδ(f)
      using Int_ZF_2_L16A Int_ZF_1_1_L4 Int_ZF_1_2_L11
        Int_triangle_ineq2 by simp
    with A2 T show abs(f(p)) ≤ A·abs(p)+B
      using Int_ZF_1_3_L14 by simp
  qed
  ultimately show thesis by auto
qed
```

If f and g are slopes, then $f \circ g$ is equivalent (almost equal) to $g \circ f$. This is Theorem 9 in Arthan's paper [2] .

```
theorem (in int1) Arthan_Th_9: assumes A1: f∈𝒮  g∈𝒮
  shows f∘g ∼ g∘f
proof -
   from A1 have
      ∃A B. A∈ℤ ∧ B∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B)
      ∃C D. C∈ℤ ∧ D∈ℤ ∧ (∀p∈ℤ. abs(g(p)) ≤ C·abs(p)+D)
      using Arthan_Lem_8 by auto
    then obtain A B C D where D1: A∈ℤ B∈ℤ C∈ℤ D∈ℤ and D2:
      ∀p∈ℤ. abs(f(p)) ≤ A·abs(p)+B
      ∀p∈ℤ. abs(g(p)) ≤ C·abs(p)+D
      by auto
    let E = maxδ(g)·(A+1) + maxδ(f)·(C+1)
    let F = (B·maxδ(g) + 2·maxδ(g)) + (D·maxδ(f) + 2·maxδ(f))
  { fix p assume A2: p∈ℤ
    with A1 have T1:
      g(p) ∈ ℤ  f(p) ∈ ℤ  abs(p) ∈ ℤ  2 ∈ ℤ
      f(g(p)) ∈ ℤ  g(f(p)) ∈ ℤ  f(g(p)) - g(f(p)) ∈ ℤ
      p·f(g(p)) ∈ ℤ  p·g(f(p)) ∈ ℤ
      abs(f(g(p))-g(f(p))) ∈ ℤ
      using Int_ZF_2_1_L2B Int_ZF_2_1_L10 Int_ZF_1_1_L5 Int_ZF_2_L14 int_two_three_are_int
      by auto
    with A1 A2 have
      abs((f(g(p))-g(f(p)))·p) ≤
      (abs(p)+abs(f(p))+2)·maxδ(g) + (abs(p)+abs(g(p))+2)·maxδ(f)
      using Arthan_Lem_7 Int_ZF_1_2_L10A Int_ZF_1_2_L12 by simp
    moreover have
      (abs(p)+abs(f(p))+2)·maxδ(g) + (abs(p)+abs(g(p))+2)·maxδ(f) ≤
      ((maxδ(g)·(A+1) + maxδ(f)·(C+1)))·abs(p) +
      ((B·maxδ(g) + 2·maxδ(g)) + (D·maxδ(f) + 2·maxδ(f)))
    proof -
      from D2 A2 T1 have
```

```
        abs(p)+abs(f(p))+2 ≤ abs(p)+(A·abs(p)+B)+2
        abs(p)+abs(g(p))+2 ≤ abs(p)+(C·abs(p)+D)+2
        using Int_ZF_2_L15C by auto
      with A1 have
        (abs(p)+abs(f(p))+2)·maxδ(g) ≤ (abs(p)+(A·abs(p)+B)+2)·maxδ(g)
        (abs(p)+abs(g(p))+2)·maxδ(f) ≤ (abs(p)+(C·abs(p)+D)+2)·maxδ(f)
        using Int_ZF_2_1_L8 Int_ZF_1_3_L13 by auto
      moreover from A1 D1 T1 have
        (abs(p)+(A·abs(p)+B)+2)·maxδ(g) =
        maxδ(g)·(A+1)·abs(p) + (B·maxδ(g) + 2·maxδ(g))
        (abs(p)+(C·abs(p)+D)+2)·maxδ(f) =
        maxδ(f)·(C+1)·abs(p) + (D·maxδ(f) + 2·maxδ(f))
        using Int_ZF_2_1_L8 Int_ZF_1_2_L13 by auto
      ultimately have
        (abs(p)+abs(f(p))+2)·maxδ(g) + (abs(p)+abs(g(p))+2)·maxδ(f) ≤
        (maxδ(g)·(A+1)·abs(p) + (B·maxδ(g) + 2·maxδ(g))) +
        (maxδ(f)·(C+1)·abs(p) + (D·maxδ(f) + 2·maxδ(f)))
        using int_ineq_add_sides by simp
      moreover from A1 A2 D1 have abs(p) ∈ ℤ
        maxδ(g)·(A+1) ∈ ℤ  B·maxδ(g) + 2·maxδ(g) ∈ ℤ
        maxδ(f)·(C+1) ∈ ℤ  D·maxδ(f) + 2·maxδ(f) ∈ ℤ
        using Int_ZF_2_L14 Int_ZF_2_1_L8 int_zero_one_are_int
          Int_ZF_1_1_L5 int_two_three_are_int by auto
      ultimately show thesis using Int_ZF_1_2_L14 by simp
    qed
    ultimately have
      abs((f(g(p))-g(f(p)))·p) ≤ E·abs(p) + F
      by (rule Int_order_transitive)
    with A2 T1 have
      abs(f(g(p))-g(f(p)))·abs(p) ≤ E·abs(p) + F
      abs(f(g(p))-g(f(p))) ∈ ℤ
      using Int_ZF_1_3_L5 by auto
  } then have
      ∀p∈ℤ. abs(f(g(p))-g(f(p))) ∈ ℤ
      ∀p∈ℤ. abs(f(g(p))-g(f(p)))·abs(p) ≤ E·abs(p) + F
    by auto
  moreover from A1 D1 have E ∈ ℤ  F ∈ ℤ
    using int_zero_one_are_int int_two_three_are_int Int_ZF_2_1_L8 Int_ZF_1_1_L5
    by auto
  ultimately have
    ∃L. ∀p∈ℤ. abs(f(g(p))-g(f(p))) ≤ L
    by (rule Int_ZF_1_7_L1)
  with A1 obtain L where ∀p∈ℤ. abs((f∘g)(p)-(g∘f)(p)) ≤ L
    using Int_ZF_2_1_L10 by auto
  moreover from A1 have f∘g ∈ 𝒮  g∘f ∈ 𝒮
    using Int_ZF_2_1_L11 by auto
  ultimately show f∘g ∼ g∘f using Int_ZF_2_1_L9 by auto
qed
```

26.3 Positive slopes

This section provides background material for defining the order relation on real numbers.

Positive slopes are functions (of course.)

```
lemma (in int1) Int_ZF_2_3_L1: assumes A1: f∈𝒮₊ shows f:ℤ→ℤ
  using prems AlmostHoms_def PositiveSet_def by simp
```

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

```
lemma (in int1) Int_ZF_2_3_L1A:
  assumes A1: f∈𝒮₊ and A2: ∃n ∈ f(ℤ₊) ∩ ℤ₊. a≤n
  shows ∃M∈ℤ₊. a ≤ f(M)
proof -
 from A1 have f:ℤ→ℤ  ℤ₊ ⊆ ℤ
    using AlmostHoms_def PositiveSet_def by auto
 with A2 show thesis using func_imagedef by auto
qed
```

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

```
lemma (in int1) Arthan_Lem_3:
  assumes A1: f∈𝒮₊ and A2: D ∈ ℤ₊
  shows ∃M∈ℤ₊. ∀m∈ℤ₊. (m+1)·D ≤ f(m·M)
proof -
  let E = maxδ(f) + D
  let A = f(ℤ₊) ∩ ℤ₊
  from A1 A2 have I: D≤E
    using Int_ZF_1_5_L3 Int_ZF_2_1_L8 Int_ZF_2_L1A Int_ZF_2_L15D
    by simp
  from A1 A2 have A ⊆ ℤ₊  A ∉ Fin(ℤ)  2·E ∈ ℤ
    using int_two_three_are_int Int_ZF_2_1_L8 PositiveSet_def Int_ZF_1_1_L5
    by auto
  with A1 have ∃M∈ℤ₊.  2·E ≤ f(M)
    using Int_ZF_1_5_L2A Int_ZF_2_3_L1A by simp
  then obtain M where II: M∈ℤ₊  and III: 2·E ≤ f(M)
    by auto
  { fix m assume m∈ℤ₊ then have A4: 1≤m
      using Int_ZF_1_5_L3 by simp
    moreover from II III have (1+1) ·E ≤ f(1·M)
      using PositiveSet_def Int_ZF_1_1_L4 by simp
    moreover have ∀k.
      1≤k ∧ (k+1)·E ≤ f(k·M) ⟶ (k+1+1)·E ≤ f((k+1)·M)
    proof -
      { fix k assume A5: 1≤k  and A6: (k+1)·E ≤ f(k·M)
        with A1 A2 II have T:
          k∈ℤ  M∈ℤ  k+1 ∈ ℤ  E∈ℤ  (k+1)·E ∈ ℤ  2·E ∈ ℤ
          using Int_ZF_2_L1A PositiveSet_def int_zero_one_are_int
            Int_ZF_1_1_L5 Int_ZF_2_1_L8 by auto
        from A1 A2 A5 II have
```

```
        δ(f,k·M,M) ∈ ℤ   abs(δ(f,k·M,M)) ≤ maxδ(f)   0≤D
        using Int_ZF_2_L1A PositiveSet_def Int_ZF_1_1_L5
          Int_ZF_2_1_L7 Int_ZF_2_L16C by auto
      with III A6 have
        (k+1)·E + (2·E - E) ≤ f(k·M) + (f(M) + δ(f,k·M,M))
        using Int_ZF_1_3_L19A int_ineq_add_sides by simp
      with A1 T have (k+1+1)·E ≤ f((k+1)·M)
        using Int_ZF_1_1_L1 int_zero_one_are_int Int_ZF_1_1_L4
          Int_ZF_1_2_L11 Int_ZF_2_1_L13 by simp
    } then show thesis by simp
  qed
  ultimately have (m+1)·E ≤ f(m·M) by (rule Induction_on_int)
  with A4 I have (m+1)·D ≤ f(m·M) using Int_ZF_1_3_L13A
    by simp
} then have ∀m∈ℤ₊.(m+1)·D ≤ f(m·M) by simp
with II show thesis by auto
qed
```

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

```
corollary (in int1) Arthan_L_3_spec: assumes A1: f ∈ 𝒮₊
  shows ∃M∈ℤ₊.∀n∈ℤ₊. n+1 ≤ f(n·M)
proof -
  have ∀n∈ℤ₊. n+1 ∈ ℤ
    using PositiveSet_def int_zero_one_are_int Int_ZF_1_1_L5
    by simp
  then have ∀n∈ℤ₊. (n+1)·1 = n+1
    using Int_ZF_1_1_L4 by simp
  moreover from A1 have ∃M∈ℤ₊. ∀n∈ℤ₊. (n+1)·1 ≤ f(n·M)
    using int_one_two_are_pos Arthan_Lem_3 by simp
  ultimately show thesis by simp
qed
```

We know from Group_ZF_3.thy that finite range functions are almost homomorphisms. Besides reminding that fact for slopes the next lemma shows that finite range functions do not belong to $\mathcal{S}_+$. This is important, because the projection of the set of finite range functions defines zero in the real number construction in Real_ZF_x.thy series, while the projection of $\mathcal{S}_+$ becomes the set of (strictly) positive reals. We don't want zero to be positive, do we? The next lemma is a part of Lemma 5 in the Arthan's paper [2].

```
lemma (in int1) Int_ZF_2_3_L1B:
  assumes A1: f ∈ FinRangeFunctions(ℤ,ℤ)
  shows f∈𝒮   f ∉ 𝒮₊
proof -
  from A1 show f∈𝒮 using Int_ZF_2_1_L1 group1.Group_ZF_3_3_L1
    by auto
  have ℤ₊ ⊆ ℤ using PositiveSet_def by auto
  with A1 have f(ℤ₊) ∈ Fin(ℤ)
    using Finite1_L21 by simp
```

```
  then have f(ℤ₊) ∩ ℤ₊ ∈ Fin(ℤ)
    using Fin_subset_lemma by blast
  thus f ∉ 𝒮₊ by auto
qed
```

We want to show that if f is a slope and neither f nor $-f$ are in $\mathcal{S}_+$, then f is bounded. The next lemma is the first step towards that goal and shows that if slope is not in $\mathcal{S}_+$ then $f(\mathbb{Z}_+)$ is bounded above.

```
lemma (in int1) Int_ZF_2_3_L2: assumes A1: f∈𝒮 and A2: f ∉ 𝒮₊
  shows IsBoundedAbove(f(ℤ₊), IntegerOrder)
proof -
  from A1 have f:ℤ→ℤ using AlmostHoms_def by simp
  then have f(ℤ₊) ⊆ ℤ using func1_1_L6 by simp
  moreover from A1 A2 have f(ℤ₊) ∩ ℤ₊ ∈ Fin(ℤ) by auto
  ultimately show thesis using Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L4
    by simp
qed
```

If f is a slope and $-f \notin \mathcal{S}_+$, then $f(\mathbb{Z}_+)$ is bounded below.

```
lemma (in int1) Int_ZF_2_3_L3: assumes A1: f∈𝒮 and A2: -f ∉ 𝒮₊
  shows IsBoundedBelow(f(ℤ₊), IntegerOrder)
proof -
  from A1 have T: f:ℤ→ℤ using AlmostHoms_def by simp
  then have (-(f(ℤ₊))) = (-f)(ℤ₊)
    using Int_ZF_1_T2 group0_2_T2 PositiveSet_def func1_1_L15C
    by auto
  with A1 A2 T show IsBoundedBelow(f(ℤ₊), IntegerOrder)
    using Int_ZF_2_1_L12 Int_ZF_2_3_L2 PositiveSet_def func1_1_L6
      Int_ZF_2_T1 group3.OrderedGroup_ZF_2_L5 by simp
qed
```

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

```
lemma (in int1) Int_ZF_2_3_L4:
  assumes A1: f∈𝒮 and A2: m∈ℤ
  and A3: ∀n∈ℤ₊. abs(f(n)) ≤ L
  shows abs(f(m)) ≤ 2·maxδ(f) + L
proof -
  from A1 A3 have
    0 ≤ abs(f(1))  abs(f(1)) ≤ L
    using int_zero_one_are_int Int_ZF_2_1_L2B int_abs_nonneg int_one_two_are_pos
    by auto
  then have II: 0≤L by (rule Int_order_transitive)
  from A2 have m∈ℤ .
  moreover have abs(f(0)) ≤ 2·maxδ(f) + L
  proof -
    from A1 have
      abs(f(0)) ≤ maxδ(f)  0 ≤ maxδ(f)
      and T: maxδ(f) ∈ ℤ
```

```
      using Int_ZF_2_1_L8 by auto
    with II have abs(f(0)) ≤ maxδ(f) + maxδ(f) + L
      using Int_ZF_2_L15F by simp
    with T show thesis using Int_ZF_1_1_L4 by simp
  qed
  moreover from A1 A3 II have
    ∀n∈ℤ₊. abs(f(n)) ≤ 2·maxδ(f) + L
    using Int_ZF_2_1_L8 Int_ZF_1_3_L5A Int_ZF_2_L15F
    by simp
  moreover have ∀n∈ℤ₊. abs(f(-n)) ≤ 2·maxδ(f) + L
  proof
    fix n assume n∈ℤ₊
    with A1 A3 have
      2·maxδ(f) ∈ ℤ
      abs(f(-n)) ≤ 2·maxδ(f) + abs(f(n))
      abs(f(n)) ≤ L
      using int_two_three_are_int Int_ZF_2_1_L8 Int_ZF_1_1_L5
        PositiveSet_def Int_ZF_2_1_L14 by auto
    then show abs(f(-n)) ≤ 2·maxδ(f) + L
      using Int_ZF_2_L15A by blast
  qed
  ultimately show thesis by (rule Int_ZF_2_L19B)
qed
```

A slope whose image of the set of positive integers is bounded is a finite range function.

```
lemma (in int1) Int_ZF_2_3_L4A:
  assumes A1: f∈𝒮 and A2: IsBounded(f(ℤ₊), IntegerOrder)
  shows f ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  have T1: ℤ₊ ⊆ ℤ using PositiveSet_def by auto
  from A1 have T2: f:ℤ→ℤ using AlmostHoms_def by simp
  from A2 obtain L where ∀a∈f(ℤ₊). abs(a) ≤ L
    using Int_ZF_1_3_L20A by auto
  with T2 T1 have ∀n∈ℤ₊. abs(f(n)) ≤ L
    by (rule func1_1_L15B)
  with A1 have ∀m∈ℤ. abs(f(m)) ≤ 2·maxδ(f) + L
    using Int_ZF_2_3_L4 by simp
  with T2 have f(ℤ) ∈ Fin(ℤ)
    by (rule Int_ZF_1_3_L20C)
  with T2 show f ∈ FinRangeFunctions(ℤ,ℤ)
    using FinRangeFunctions_def by simp
qed
```

A slope whose image of the set of positive integers is bounded below is a finite range function or a positive slope.

```
lemma (in int1) Int_ZF_2_3_L4B:
  assumes f∈𝒮 and IsBoundedBelow(f(ℤ₊), IntegerOrder)
  shows f ∈ FinRangeFunctions(ℤ,ℤ) ∨ f∈𝒮₊
```

```
  using prems Int_ZF_2_3_L2 IsBounded_def Int_ZF_2_3_L4A
  by auto
```

If one slope is not greater then another on positive integers, then they are almost equal or the difference is a positive slope.

```
lemma (in int1) Int_ZF_2_3_L4C: assumes A1: f∈𝒮  g∈𝒮 and
  A2: ∀n∈ℤ₊. f(n) ≤ g(n)
  shows f∼g ∨ g + (-f) ∈ 𝒮₊
proof -
  let h = g + (-f)
  from A1 have (-f) ∈ 𝒮 using Int_ZF_2_1_L12
    by simp
  with A1 have I: h ∈ 𝒮 using Int_ZF_2_1_L12C
    by simp
  moreover have IsBoundedBelow(h(ℤ₊), IntegerOrder)
  proof -
    from I have
      h:ℤ→ℤ and ℤ₊⊆ℤ using AlmostHoms_def PositiveSet_def
      by auto
    moreover from A1 A2 have ∀n∈ℤ₊. ⟨0, h(n)⟩ ∈ IntegerOrder
      using Int_ZF_2_1_L2B PositiveSet_def Int_ZF_1_3_L10A
        Int_ZF_2_1_L12 Int_ZF_2_1_L12B Int_ZF_2_1_L12A
      by simp
    ultimately show IsBoundedBelow(h(ℤ₊), IntegerOrder)
      by (rule func_ZF_8_L1)
  qed
  ultimately have h ∈ FinRangeFunctions(ℤ,ℤ) ∨ h∈𝒮₊
    using Int_ZF_2_3_L4B by simp
  with A1 show f∼g ∨ g + (-f) ∈ 𝒮₊
    using Int_ZF_2_1_L9C by auto
qed
```

Positive slopes are arbitrarily large for large enough arguments.

```
lemma (in int1) Int_ZF_2_3_L5:
  assumes A1: f∈𝒮₊ and A2: K∈ℤ
  shows ∃N∈ℤ₊. ∀m. N≤m ⟶ K ≤ f(m)
proof -
  from A1 obtain M where I: M∈ℤ₊ and II: ∀n∈ℤ₊. n+1 ≤ f(n·M)
    using Arthan_L_3_spec by auto
  let j = GreaterOf(IntegerOrder,M,K - (minf(f,0..(M-1)) - maxδ(f)) -
1)
  from A1 I have T1:
    minf(f,0..(M-1)) - maxδ(f) ∈ ℤ  M∈ℤ
    using Int_ZF_2_1_L15 Int_ZF_2_1_L8 Int_ZF_1_1_L5 PositiveSet_def
    by auto
  with A2 I have T2:
    K - (minf(f,0..(M-1)) - maxδ(f)) ∈ ℤ
    K - (minf(f,0..(M-1)) - maxδ(f)) - 1 ∈ ℤ
    using Int_ZF_1_1_L5 int_zero_one_are_int by auto
```

```
  with T1 have III: M ≤ j  and
    K - (minf(f,0..(M-1)) - maxδ(f)) - 1 ≤ j
    using Int_ZF_1_3_L18 by auto
  with A2 T1 T2 have
    IV: K ≤ j+1 + (minf(f,0..(M-1)) - maxδ(f))
    using int_zero_one_are_int Int_ZF_2_L9C by simp
  let N = GreaterOf(IntegerOrder,1,j·M)
  from T1 III have T3: j ∈ ℤ  j·M ∈ ℤ
    using Int_ZF_2_L1A Int_ZF_1_1_L5 by auto
  then have V: N ∈ ℤ+ and VI: j·M ≤ N
    using int_zero_one_are_int Int_ZF_1_5_L3 Int_ZF_1_3_L18
    by auto
  { fix m
    let n = m zdiv M
    let k = m zmod M
    assume N≤m
    with VI have j·M ≤ m by (rule Int_order_transitive)
    with I III have
      VII: m = n·M+k
      j ≤ n  and
      VIII: n ∈ ℤ+  k ∈ 0..(M-1)
      using IntDiv_ZF_1_L5 by auto
    with II have
      j + 1 ≤ n + 1  n+1 ≤ f(n·M)
      using int_zero_one_are_int int_ord_transl_inv by auto
    then have j + 1 ≤  f(n·M)
      by (rule Int_order_transitive)
    with T1 have
      j+1 + (minf(f,0..(M-1)) - maxδ(f)) ≤
      f(n·M) + (minf(f,0..(M-1)) - maxδ(f))
      using int_ord_transl_inv by simp
    with IV have K ≤ f(n·M) + (minf(f,0..(M-1)) - maxδ(f))
      by (rule Int_order_transitive)
    moreover from A1 I VIII have
      f(n·M) + (minf(f,0..(M-1))- maxδ(f)) ≤ f(n·M+k)
      using PositiveSet_def Int_ZF_2_1_L16 by simp
    ultimately have K ≤ f(n·M+k)
      by (rule Int_order_transitive)
    with VII have K ≤ f(m) by simp
    } then have  ∀m. N≤m ⟶ K ≤ f(m)
      by simp
    with V show thesis by auto
qed
```

Positive slopes are arbitrarily small for small enough arguments. Kind of dual to Int_ZF_2_3_L5.

```
lemma (in int1) Int_ZF_2_3_L5A: assumes A1: f∈S+ and A2: K∈ℤ
  shows ∃N∈ℤ+. ∀m. N≤m ⟶ f(-m) ≤ K
proof -
```

```
  from A1 have T1: abs(f(0)) + maxδ(f) ∈ ℤ
    using Int_ZF_2_1_L8 by auto
  with A2 have abs(f(0)) + maxδ(f) - K ∈ ℤ
    using Int_ZF_1_1_L5 by simp
  with A1 have
    ∃N∈ℤ+. ∀m. N≤m ⟶ abs(f(0)) + maxδ(f) - K ≤ f(m)
    using Int_ZF_2_3_L5 by simp
  then obtain N where I: N∈ℤ+ and II:
    ∀m. N≤m ⟶  abs(f(0)) + maxδ(f) - K ≤ f(m)
    by auto
  { fix m assume A3: N≤m
    with A1 have
      f(-m) ≤ abs(f(0)) + maxδ(f) - f(m)
      using Int_ZF_2_L1A Int_ZF_2_1_L14 by simp
    moreover
    from II T1 A3 have abs(f(0)) + maxδ(f) - f(m) ≤
      (abs(f(0)) + maxδ(f)) -(abs(f(0)) + maxδ(f) - K)
      using Int_ZF_2_L10 int_ord_transl_inv by simp
    with A2 T1 have abs(f(0)) + maxδ(f) - f(m) ≤ K
      using Int_ZF_1_2_L3 by simp
    ultimately have f(-m) ≤ K
      by (rule Int_order_transitive)
  } then have ∀m. N≤m  ⟶ f(-m) ≤ K
    by simp
  with I show thesis by auto
qed
```

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

```
corollary (in int1) Int_ZF_2_3_L6: assumes f∈𝒮+
  shows ∃N∈ℤ+. ∀m. N≤m ⟶ f(m) ∈ ℤ+
  using prems int_zero_one_are_int Int_ZF_2_3_L5 Int_ZF_1_5_L3
  by simp
```

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

```
corollary (in int1) Int_ZF_2_3_L6A: assumes f∈𝒮+ and K∈ℤ
   shows ∃N∈ℤ+. K ≤ f(N)
proof -
  from prems have ∃N∈ℤ+. ∀m. N≤m ⟶ K ≤ f(m)
    using Int_ZF_2_3_L5 by simp
  then obtain N where I: N ∈ ℤ+  and II: ∀m. N≤m ⟶ K ≤ f(m)
    by auto
  then show thesis using PositiveSet_def int_ord_is_refl refl_def
    by auto
qed
```

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

```
lemma (in int1) Int_ZF_2_3_L7: assumes A1: f∈𝒮
  and A2: ∀K∈ℤ. ∃n∈ℤ+. K ≤ f(n)
  shows f ∈ 𝒮+
```

```
proof -
  { fix K assume K∈ℤ
    with A2 obtain n where n∈ℤ₊  K ≤ f(n)
      by auto
    moreover from A1 have ℤ₊ ⊆ ℤ  f:ℤ→ℤ
      using PositiveSet_def AlmostHoms_def by auto
    ultimately have ∃m ∈ f(ℤ₊). K ≤ m
      using func1_1_L15D by auto
  } then have ∀K∈ℤ. ∃m ∈ f(ℤ₊). K ≤ m by simp
  with A1 show f ∈ 𝒮₊ using Int_ZF_4_L9 Int_ZF_2_3_L2
    by auto
qed
```

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

```
theorem (in int1) Int_ZF_2_3_L8:
  assumes A1: f∈𝒮 and A2: f ∉ FinRangeFunctions(ℤ,ℤ)
  shows (f ∈ 𝒮₊) Xor ((-f) ∈ 𝒮₊)
proof -
  have T1: ℤ₊ ⊆ ℤ using PositiveSet_def by auto
  from A1 have T2: f:ℤ→ℤ  using AlmostHoms_def by simp
  then have I: f(ℤ₊) ⊆ ℤ using func1_1_L6 by auto
  from A1 A2 have f ∈ 𝒮₊ ∨ (-f) ∈ 𝒮₊
    using Int_ZF_2_3_L2 Int_ZF_2_3_L3 IsBounded_def Int_ZF_2_3_L4A
    by auto
  moreover have ¬(f ∈ 𝒮₊ ∧ (-f) ∈ 𝒮₊)
  proof -
    { assume A3: f ∈ 𝒮₊  and A4: (-f) ∈ 𝒮₊
      from A3 obtain N1 where
        I: N1∈ℤ₊ and II: ∀m. N1≤m ⟶ f(m) ∈ ℤ₊
        using Int_ZF_2_3_L6 by auto
      from A4 obtain N2 where
        III: N2∈ℤ₊ and IV: ∀m. N2≤m ⟶ (-f)(m) ∈ ℤ₊
        using Int_ZF_2_3_L6 by auto
      let N = GreaterOf(IntegerOrder,N1,N2)
      from I III have N1 ≤ N  N2 ≤ N
        using PositiveSet_def Int_ZF_1_3_L18 by auto
      with A1 II IV have
        f(N) ∈ ℤ₊  (-f)(N) ∈ ℤ₊  (-f)(N) = -(f(N))
        using Int_ZF_2_L1A PositiveSet_def Int_ZF_2_1_L12A
        by auto
      then have False using Int_ZF_1_5_L8 by simp
    } thus thesis by auto
  qed
  ultimately show (f ∈ 𝒮₊) Xor ((-f) ∈ 𝒮₊)
    using Xor_def by simp
qed
```

The sum of positive slopes is a positive slope.

```
theorem (in int1) sum_of_pos_sls_is_pos_sl:
```

```
  assumes A1: f ∈ 𝒮₊  g ∈ 𝒮₊
  shows f+g ∈ 𝒮₊
proof -
  { fix K assume K∈ℤ
    with A1 have ∃N∈ℤ₊. ∀m. N≤m ⟶ K ≤ f(m)
      using Int_ZF_2_3_L5 by simp
    then obtain N where I: N∈ℤ₊ and II: ∀m. N≤m ⟶ K ≤ f(m)
      by auto
    from A1 have ∃M∈ℤ₊. ∀m. M≤m ⟶ 0 ≤ g(m)
      using int_zero_one_are_int Int_ZF_2_3_L5 by simp
    then obtain M where III: M∈ℤ₊ and IV: ∀m. M≤m ⟶ 0 ≤ g(m)
      by auto
    let L = GreaterOf(IntegerOrder,N,M)
    from I III have V: L ∈ ℤ₊  ℤ₊ ⊆ ℤ
      using GreaterOf_def PositiveSet_def by auto
    moreover from A1 V have (f+g)(L) = f(L) + g(L)
      using Int_ZF_2_1_L12B by auto
    moreover from I II III IV have K ≤ f(L) + g(L)
      using PositiveSet_def Int_ZF_1_3_L18 Int_ZF_2_L15F
      by simp
    ultimately have L ∈ ℤ₊  K ≤ (f+g)(L)
      by auto
    then have ∃n ∈ℤ₊. K ≤ (f+g)(n)
      by auto
  } with A1 show f+g ∈ 𝒮₊
    using Int_ZF_2_1_L12C Int_ZF_2_3_L7 by simp
qed
```

The composition of positive slopes is a positive slope.

```
theorem (in int1) comp_of_pos_sls_is_pos_sl:
  assumes A1: f ∈ 𝒮₊  g ∈ 𝒮₊
  shows f∘g ∈ 𝒮₊
proof -
  { fix K assume K∈ℤ
    with A1 have ∃N∈ℤ₊. ∀m. N≤m ⟶ K ≤ f(m)
      using Int_ZF_2_3_L5 by simp
    then obtain N where N∈ℤ₊ and I: ∀m. N≤m ⟶ K ≤ f(m)
      by auto
    with A1 have ∃M∈ℤ₊. N ≤ g(M)
      using PositiveSet_def Int_ZF_2_3_L6A by simp
    then obtain M where M∈ℤ₊  N ≤ g(M)
      by auto
    with A1 I have ∃M∈ℤ₊. K ≤ (f∘g)(M)
      using PositiveSet_def Int_ZF_2_1_L10
      by auto
  } with A1 show f∘g ∈ 𝒮₊
    using Int_ZF_2_1_L11 Int_ZF_2_3_L7
    by simp
qed
```

A slope equivalent to a positive one is positive.

```
lemma (in int1) Int_ZF_2_3_L9:
  assumes A1: f ∈ 𝒮+ and A2: ⟨f,g⟩ ∈ AlEqRel shows g ∈ 𝒮+
proof -
  from A2 have T: g∈𝒮 and ∃L∈ℤ. ∀m∈ℤ. abs(f(m)-g(m)) ≤ L
    using Int_ZF_2_1_L9A by auto
   then obtain L where
     I: L∈ℤ  and II: ∀m∈ℤ. abs(f(m)-g(m)) ≤ L
     by auto
  { fix K assume A3: K∈ℤ
    with I have K+L ∈ ℤ
      using Int_ZF_1_1_L5 by simp
    with A1 obtain M where III: M∈ℤ+  and IV: K+L ≤ f(M)
      using Int_ZF_2_3_L6A by auto
    with A1 A3 I have  K ≤ f(M)-L
      using PositiveSet_def Int_ZF_2_1_L2B Int_ZF_2_L9B
      by simp
    moreover from A1 T II III have
      f(M)-L ≤ g(M)
      using PositiveSet_def Int_ZF_2_1_L2B Int_triangle_ineq2
      by simp
    ultimately have K ≤  g(M)
      by (rule Int_order_transitive)
    with III have ∃n∈ℤ+. K ≤ g(n)
      by auto
  } with T show g ∈ 𝒮+
    using Int_ZF_2_3_L7 by simp
qed
```

The set of positive slopes is saturated with respect to the relation of equivalence of slopes.

```
lemma (in int1) pos_slopes_saturated: shows IsSaturated(AlEqRel,𝒮+)
proof -
  have
    equiv(𝒮,AlEqRel)
    AlEqRel ⊆ 𝒮 × 𝒮
    using Int_ZF_2_1_L9B by auto
  moreover have 𝒮+ ⊆ 𝒮 by auto
  moreover have ∀f∈𝒮+. ∀g∈𝒮. ⟨f,g⟩ ∈ AlEqRel ⟶ g ∈ 𝒮+
    using Int_ZF_2_3_L9 by blast
  ultimately show IsSaturated(AlEqRel,𝒮+)
    by (rule EquivClass_3_L3)
qed
```

A technical lemma involving a projection of the set of positive slopes and a logical epression with exclusive or.

```
lemma (in int1) Int_ZF_2_3_L10:
  assumes A1: f∈𝒮  g∈𝒮
```

```
  and A2: R = {AlEqRel{s}. s∈𝒮₊}
  and A3: (f∈𝒮₊) Xor (g∈𝒮₊)
  shows (AlEqRel{f} ∈ R) Xor (AlEqRel{g} ∈ R)
proof -
  from A1 A2 A3 have
    equiv(𝒮,AlEqRel)
    IsSaturated(AlEqRel,𝒮₊)
    𝒮₊ ⊆ 𝒮
    f∈𝒮  g∈𝒮
    R = {AlEqRel{s}. s∈𝒮₊}
    (f∈𝒮₊) Xor (g∈𝒮₊)
    using pos_slopes_saturated Int_ZF_2_1_L9B by auto
  then show thesis by (rule EquivClass_3_L7)
qed
```

Identity function is a positive slope.

```
lemma (in int1) Int_ZF_2_3_L11: shows id(ℤ) ∈ 𝒮₊
proof -
  let f = id(ℤ)
  { fix K assume K∈ℤ
    then obtain n where T: n∈ℤ₊ and K≤n
      using Int_ZF_1_5_L9 by auto
    moreover from T have f(n) = n
      using PositiveSet_def by simp
    ultimately have  n∈ℤ₊ and K≤f(n)
      by auto
    then have ∃n∈ℤ₊. K≤f(n) by auto
  } then show f ∈ 𝒮₊
    using Int_ZF_2_1_L17 Int_ZF_2_3_L7 by simp
qed
```

The identity function is not almost equal to any bounded function.

```
lemma (in int1) Int_ZF_2_3_L12: assumes A1: f ∈ FinRangeFunctions(ℤ,ℤ)
  shows ¬(id(ℤ) ∼ f)
proof -
  { from A1 have id(ℤ) ∈ 𝒮₊
      using Int_ZF_2_3_L11 by simp
    moreover assume ⟨id(ℤ),f⟩ ∈ AlEqRel
    ultimately have  f ∈ 𝒮₊
      by (rule Int_ZF_2_3_L9)
    with A1 have False using Int_ZF_2_3_L1B
      by simp
  } then show ¬(id(ℤ) ∼ f) by auto
qed
```

26.4 Inverting slopes

Not every slope is a 1:1 function. However, we can still invert slopes in the sense that if f is a slope, then we can find a slope g such that $f \circ g$ is almost equal to the identity function. The goal of this this section is to establish this fact for positive slopes.

If f is a positive slope, then for every positive integer p the set $\{n \in Z_+ : p \leq f(n)\}$ is a nonempty subset of positive integers. Recall that $f^{-1}(p)$ is the notation for the smallest element of this set.

```
lemma (in int1) Int_ZF_2_4_L1:
  assumes A1: f ∈ S+ and A2: p∈ℤ+ and A3: A = {n∈ℤ+. p ≤ f(n)}
  shows
  A ⊆ ℤ+
  A ≠ 0
  f⁻¹(p) ∈ A
  ∀m∈A. f⁻¹(p) ≤ m
proof -
  from A3 show I: A ⊆ ℤ+ by auto
  from A1 A2 have ∃n∈ℤ+. p ≤ f(n)
    using PositiveSet_def Int_ZF_2_3_L6A by simp
  with A3 show II: A ≠ 0 by auto
  from A3 I II show
    f⁻¹(p) ∈ A
    ∀m∈A. f⁻¹(p) ≤ m
    using Int_ZF_1_5_L1C by auto
qed
```

If f is a positive slope and p is a positive integer p, then $f^{-1}(p)$ (defined as the minimum of the set $\{n \in Z_+ : p \leq f(n)\}$) is a (well defined) positive integer.

```
lemma (in int1) Int_ZF_2_4_L2:
  assumes f ∈ S+ and p∈ℤ+
  shows
  f⁻¹(p) ∈ ℤ+
  p ≤ f(f⁻¹(p))
  using prems Int_ZF_2_4_L1 by auto
```

If f is a positive slope and p is a positive integer such that $n \leq f(p)$, then $f^{-1}(n) \leq p$.

```
lemma (in int1) Int_ZF_2_4_L3:
  assumes f ∈ S+ and  m∈ℤ+  p∈ℤ+ and m ≤ f(p)
  shows f⁻¹(m) ≤ p
  using prems Int_ZF_2_4_L1 by simp
```

An upper bound $f(f^{-1}(m) - 1)$ for positive slopes.

```
lemma (in int1) Int_ZF_2_4_L4:
  assumes A1: f ∈ S+ and A2: m∈ℤ+ and A3: f⁻¹(m)-1 ∈ ℤ+
  shows f(f⁻¹(m)-1) ≤ m   f(f⁻¹(m)-1) ≠ m
proof -
  from A1 A2 have T: f⁻¹(m) ∈ ℤ using Int_ZF_2_4_L2 PositiveSet_def
    by simp
```

```
  from A1 A3 have f:ℤ→ℤ  and f⁻¹(m)-1 ∈ ℤ
    using Int_ZF_2_3_L1 PositiveSet_def by auto
  with A1 A2 have T1: f(f⁻¹(m)-1) ∈ ℤ  m∈ℤ
    using apply_funtype PositiveSet_def by auto
  { assume m ≤ f(f⁻¹(m)-1)
    with A1 A2 A3 have f⁻¹(m) ≤ f⁻¹(m)-1
      by (rule Int_ZF_2_4_L3)
    with T have False using Int_ZF_1_2_L3AA
      by simp
  } then have I: ¬(m ≤ f(f⁻¹(m)-1)) by auto
  with T1 show f(f⁻¹(m)-1) ≤ m
    by (rule Int_ZF_2_L19)
  from T1 I show f(f⁻¹(m)-1) ≠ m
    by (rule Int_ZF_2_L19)
qed
```

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

```
lemma (in int1) Int_ZF_2_4_L5:
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊ and A3: m≤n
  shows f⁻¹(m) ≤ f⁻¹(n)
proof -
  from A2 A3 have T: n ∈ ℤ₊ using Int_ZF_1_5_L7 by blast
  with A1 have n ≤ f(f⁻¹(n)) using Int_ZF_2_4_L2
    by simp
  with A3 have m ≤ f(f⁻¹(n)) by (rule Int_order_transitive)
  with A1 A2 T show f⁻¹(m) ≤ f⁻¹(n)
    using Int_ZF_2_4_L2 Int_ZF_2_4_L3 by simp
qed
```

If $f^{-1}(m)$ is positive and n is a positive integer, then, then $f^{-1}(m+n)-1$ is positive.

```
lemma (in int1) Int_ZF_2_4_L6:
  assumes A1: f ∈ 𝒮₊ and A2: m∈ℤ₊  n∈ℤ₊ and
  A3: f⁻¹(m)-1 ∈ ℤ₊
  shows f⁻¹(m+n)-1 ∈ ℤ₊
proof -
  from A1 A2 have f⁻¹(m)-1 ≤  f⁻¹(m+n) - 1
     using PositiveSet_def Int_ZF_1_5_L7A Int_ZF_2_4_L2
       Int_ZF_2_4_L5 int_zero_one_are_int Int_ZF_1_1_L4
       int_ord_transl_inv by simp
  with A3 show f⁻¹(m+n)-1 ∈ ℤ₊ using Int_ZF_1_5_L7
    by blast
qed
```

If f is a slope, then $f(f^{-1}(m+n)-f^{-1}(m)-f^{-1}(n))$ is uniformly bounded above and below. Will it be the messiest IsarMathLib proof ever? Only time will tell.

```
lemma (in int1) Int_ZF_2_4_L7:  assumes A1: f ∈ 𝒮₊ and
```

```
  A2: ∀m∈ℤ+. f⁻¹(m)-1 ∈ ℤ+
  shows
  ∃U∈ℤ. ∀m∈ℤ+. ∀n∈ℤ+. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
  ∃N∈ℤ. ∀m∈ℤ+. ∀n∈ℤ+. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
proof -
  from A1 have ∃L∈ℤ. ∀r∈ℤ. f(r) ≤ f(r-1) + L
    using Int_ZF_2_1_L28 by simp
  then obtain L where
    I: L∈ℤ and II: ∀r∈ℤ. f(r) ≤ f(r-1) + L
    by auto
  from A1 have
    ∃M∈ℤ. ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    ∃K∈ℤ. ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r)-f(p)-f(q)+K ≤ f(r-p-q)
    using Int_ZF_2_1_L30 by auto
  then obtain M K where III: M∈ℤ and
    IV: ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    and
    V: K∈ℤ and VI: ∀r∈ℤ.∀p∈ℤ.∀q∈ℤ. f(r)-f(p)-f(q)+K ≤ f(r-p-q)
    by auto
  from I III V have
    L+M ∈ ℤ  (-L) - L + K ∈ ℤ
    using Int_ZF_1_1_L4 Int_ZF_1_1_L5 by auto
  moreover
    { fix m n
      assume A3: m∈ℤ+ n∈ℤ+
      have  f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤  L+M ∧
        (-L)-L+K ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
      proof -
        let r = f⁻¹(m+n)
        let p = f⁻¹(m)
        let q = f⁻¹(n)
        from A1 A3 have T1:
          p ∈ ℤ+  q ∈ ℤ+  r ∈ ℤ+
          using Int_ZF_2_4_L2 pos_int_closed_add_unfolded by auto
        with A3 have T2:
          m ∈ ℤ  n ∈ ℤ  p ∈ ℤ  q ∈ ℤ  r ∈ ℤ
          using PositiveSet_def by auto
        from A2 A3 have T3:
          r-1 ∈ ℤ+ p-1 ∈ ℤ+  q-1 ∈ ℤ+
          using pos_int_closed_add_unfolded by auto
        from A1 A3 have VII:
          m+n ≤ f(r)
          m ≤ f(p)
          n ≤ f(q)
          using Int_ZF_2_4_L2 pos_int_closed_add_unfolded by auto
        from A1 A3 T3 have VIII:
          f(r-1) ≤ m+n
          f(p-1) ≤ m
          f(q-1) ≤ n
```

```
  using pos_int_closed_add_unfolded Int_ZF_2_4_L4 by auto
have f(r-p-q) ≤ L+M
proof -
  from IV T2 have f(r-p-q) ≤ f(r)-f(p)-f(q)+M
    by simp
  moreover
  from I II T2 VIII have
    f(r) ≤ f(r-1) + L
    f(r-1) + L ≤ m+n+L
    using int_ord_transl_inv by auto
  then have f(r) ≤  m+n+L
    by (rule Int_order_transitive)
  with VII have f(r) - f(p) ≤ m+n+L-m
    using int_ineq_add_sides by simp
  with I T2 VII have f(r) - f(p) - f(q) ≤ n+L-n
    using Int_ZF_1_2_L9 int_ineq_add_sides by simp
  with I III T2 have f(r) - f(p) - f(q) + M ≤ L+M
    using Int_ZF_1_2_L3 int_ord_transl_inv by simp
  ultimately show f(r-p-q) ≤ L+M
    by (rule Int_order_transitive)
qed
moreover have (-L)-L +K ≤ f(r-p-q)
proof -
  from I II T2 VIII have
    f(p) ≤ f(p-1) + L
    f(p-1) + L ≤ m +L
    using int_ord_transl_inv by auto
  then have f(p) ≤  m +L
    by (rule Int_order_transitive)
  with VII have m+n -(m+L) ≤ f(r) - f(p)
    using int_ineq_add_sides by simp
  with I T2 have n - L ≤  f(r) - f(p)
    using Int_ZF_1_2_L9 by simp
  moreover
  from I II T2 VIII have
    f(q) ≤ f(q-1) + L
    f(q-1) + L ≤ n +L
    using int_ord_transl_inv by auto
  then have f(q) ≤  n +L
    by (rule Int_order_transitive)
  ultimately have
    n - L - (n+L) ≤  f(r) - f(p) - f(q)
    using int_ineq_add_sides by simp
  with I V T2 have
    (-L)-L +K ≤  f(r) - f(p) - f(q) + K
    using Int_ZF_1_2_L3 int_ord_transl_inv by simp
  moreover from VI T2 have
    f(r) - f(p) - f(q) + K ≤ f(r-p-q)
    by simp
```

```
          ultimately show (-L)-L +K ≤ f(r-p-q)
            by (rule Int_order_transitive)
        qed
        ultimately show
          f(r-p-q) ≤  L+M ∧
          (-L)-L+K ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
          by simp
      qed
    }
  ultimately show
    ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    by auto
qed
```

The expression $f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$ is uniformly bounded for all pairs $\langle m, n\rangle \in \mathbb{Z}_+ \times \mathbb{Z}_+$. Recall that in the `int1` context `ε(f,x)` is defined so that $\varepsilon(f, \langle m, n\rangle) = f^{-1}(m+n) - f^{-1}(m) - f^{-1}(n)$.

```
lemma (in int1) Int_ZF_2_4_L8:  assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-1 ∈ ℤ₊
  shows ∃M. ∀x∈ℤ₊×ℤ₊. abs(ε(f,x)) ≤ M
proof -
  from A1 A2 have
    ∃U∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∃N∈ℤ. ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    using  Int_ZF_2_4_L7 by auto
  then obtain U N where I:
    ∀m∈ℤ₊. ∀n∈ℤ₊. f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n)) ≤ U
    ∀m∈ℤ₊. ∀n∈ℤ₊. N ≤ f(f⁻¹(m+n)-f⁻¹(m)-f⁻¹(n))
    by auto
  have ℤ₊×ℤ₊ ≠ 0 using int_one_two_are_pos by auto
  moreover from A1 have f: ℤ→ℤ
    using AlmostHoms_def by simp
  moreover from A1 have
    ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x)
    using Int_ZF_2_3_L5 by simp
  moreover from A1 have
    ∀a∈ℤ.∃b∈ℤ₊.∀y. b≤y ⟶ f(-y) ≤ a
    using Int_ZF_2_3_L5A by simp
  moreover have
    ∀x∈ℤ₊×ℤ₊. ε(f,x) ∈ ℤ ∧ f(ε(f,x)) ≤ U ∧ N ≤ f(ε(f,x))
  proof -
    { fix x assume A3: x ∈ ℤ₊×ℤ₊
      let m = fst(x)
      let n = snd(x)
      from A3 have T: m ∈ ℤ₊  n ∈ ℤ₊  m+n ∈ ℤ₊
        using pos_int_closed_add_unfolded by auto
      with A1 have
        f⁻¹(m+n) ∈ ℤ  f⁻¹(m) ∈ ℤ  f⁻¹(n) ∈ ℤ
```

```
      using Int_ZF_2_4_L2 PositiveSet_def by auto
    with I T have
      ε(f,x) ∈ ℤ ∧ f(ε(f,x)) ≤ U ∧ N ≤ f(ε(f,x))
      using Int_ZF_1_1_L5 by auto
  } thus thesis by simp
  qed
  ultimately show ∃M.∀x∈ℤ+×ℤ+. abs(ε(f,x)) ≤ M
    by (rule Int_ZF_1_6_L4)
qed
```

The (candidate for) inverse of a positive slope is a (well defined) function on $\mathbb{Z}_+$.

```
lemma (in int1) Int_ZF_2_4_L9:
  assumes A1: f ∈ S+ and A2: g = {⟨p,f⁻¹(p)⟩. p∈ℤ+}
  shows
  g : ℤ+→ℤ+
  g : ℤ+→ℤ
proof -
  from A1 have
    ∀p∈ℤ+. f⁻¹(p) ∈ ℤ+
    ∀p∈ℤ+. f⁻¹(p) ∈ ℤ
    using Int_ZF_2_4_L2 PositiveSet_def by auto
  with A2 show
    g : ℤ+→ℤ+  and  g : ℤ+→ℤ
    using ZF_fun_from_total by auto
qed
```

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

```
lemma (in int1) Int_ZF_2_4_L10:
  assumes A1: f ∈ S+ and A2: g = {⟨p,f⁻¹(p)⟩. p∈ℤ+} and A3: p∈ℤ+
  shows g(p) = f⁻¹(p)
proof -
  from A1 A2 have  g : ℤ+→ℤ+ using Int_ZF_2_4_L9 by simp
  with A2 A3 show g(p) = f⁻¹(p) using ZF_fun_from_tot_val by simp
qed
```

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

```
lemma (in int1) Int_ZF_2_4_L11: assumes A1: f ∈ S+ and
  A2: ∀m∈ℤ+. f⁻¹(m)-1 ∈ ℤ+ and
  A3: g = {⟨p,f⁻¹(p)⟩. p∈ℤ+}
  shows OddExtension(ℤ,IntegerAddition,IntegerOrder,g) ∈ S
proof -
  from A1 A2 have ∃L. ∀x∈ℤ+×ℤ+. abs(ε(f,x)) ≤ L
    using Int_ZF_2_4_L8 by simp
  then obtain L where I: ∀x∈ℤ+×ℤ+. abs(ε(f,x)) ≤ L
    by auto
  from A1 A3 have g : ℤ+→ℤ using Int_ZF_2_4_L9
    by simp
  moreover have ∀m∈ℤ+. ∀n∈ℤ+. abs(δ(g,m,n)) ≤ L
```

```
  proof-
    { fix m n
      assume A4: m∈ℤ₊  n∈ℤ₊
      then have ⟨m,n⟩ ∈ ℤ₊×ℤ₊ by simp
      with I have abs(ε(f,⟨m,n⟩)) ≤ L by simp
      moreover have ε(f,⟨m,n⟩) = f⁻¹(m+n) - f⁻¹(m) - f⁻¹(n)
        by simp
      moreover from A1 A3 A4 have
        f⁻¹(m+n) = g(m+n)  f⁻¹(m) = g(m)  f⁻¹(n) = g(n)
        using pos_int_closed_add_unfolded Int_ZF_2_4_L10 by auto
      ultimately have abs(δ(g,m,n)) ≤ L by simp
    } thus ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ L by simp
  qed
  ultimately show thesis by (rule Int_ZF_2_1_L24)
qed
```

Every positive slope that is at least 2 on positive integers almost has an inverse.

```
lemma (in int1) Int_ZF_2_4_L12: assumes A1: f ∈ 𝒮₊ and
  A2: ∀m∈ℤ₊. f⁻¹(m)-1 ∈ ℤ₊
  shows ∃h∈𝒮. f∘h ∼ id(ℤ)
proof -
  let g = {⟨p,f⁻¹(p)⟩. p∈ℤ₊}
  let h = OddExtension(ℤ,IntegerAddition,IntegerOrder,g)
  from A1 have
    ∃M∈ℤ. ∀n∈ℤ. f(n) ≤ f(n-1) + M
    using Int_ZF_2_1_L28 by simp
  then obtain M where
    I: M∈ℤ and II: ∀n∈ℤ. f(n) ≤ f(n-1) + M
    by auto
  from A1 A2 have T: h ∈ 𝒮
    using Int_ZF_2_4_L11 by simp
  moreover have  f∘h ∼ id(ℤ)
  proof -
    from A1 T have f∘h ∈ 𝒮 using Int_ZF_2_1_L11
      by simp
    moreover note I
    moreover
    { fix m assume A3: m∈ℤ₊
      with A1 have f⁻¹(m) ∈ ℤ
        using Int_ZF_2_4_L2 PositiveSet_def by simp
      with II have f(f⁻¹(m)) ≤ f(f⁻¹(m)-1) + M
        by simp
      moreover from A1 A2 I A3 have f(f⁻¹(m)-1) + M ≤ m+M
        using Int_ZF_2_4_L4 int_ord_transl_inv by simp
      ultimately have f(f⁻¹(m)) ≤ m+M
        by (rule Int_order_transitive)
      moreover from A1 A3 have m ≤ f(f⁻¹(m))
        using Int_ZF_2_4_L2 by simp
```

```
      moreover from A1 A2 T A3 have f(f⁻¹(m)) = (f∘h)(m)
        using Int_ZF_2_4_L9 Int_ZF_1_5_L11
          Int_ZF_2_4_L10 PositiveSet_def Int_ZF_2_1_L10
        by simp
      ultimately have m ≤ (f∘h)(m) ∧ (f∘h)(m) ≤ m+M
        by simp }
    ultimately show f∘h ∼ id(ℤ) using Int_ZF_2_1_L32
      by simp
  qed
  ultimately show ∃h∈𝒮. f∘h ∼ id(ℤ)
    by auto
qed
```

Int_ZF_2_4_L12 is almost what we need, except that it has an assumption that the values of the slope that we get the inverse for are not smaller than 2 on positive integers. The Arthan's proof of Theorem 11 has a mistake where he says "note that for all but finitely many $m, n \in N$ $p = g(m)$ and $q = g(n)$ are both positive". Of course there may be infinitely many pairs $\langle m, n\rangle$ such that p, q are not both positive. This is however easy to workaround: we just modify the slope by adding a constant so that the slope is large enough on positive integers and then look for the inverse.

```
theorem (in int1) pos_slope_has_inv: assumes A1: f ∈ 𝒮₊
  shows ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(ℤ))
proof -
  from A1 have f: ℤ→ℤ  1∈ℤ  2 ∈ ℤ
    using AlmostHoms_def int_zero_one_are_int int_two_three_are_int
    by auto
  moreover from A1 have
     ∀a∈ℤ.∃b∈ℤ₊.∀x. b≤x ⟶ a ≤ f(x)
    using Int_ZF_2_3_L5 by simp
  ultimately have
    ∃c∈ℤ. 2 ≤ Minimum(IntegerOrder,{n∈ℤ₊. 1 ≤ f(n)+c})
    by (rule Int_ZF_1_6_L7)
  then obtain c where I: c∈ℤ and
    II: 2 ≤ Minimum(IntegerOrder,{n∈ℤ₊. 1 ≤ f(n)+c})
    by auto
  let g = {⟨m,f(m)+c⟩. m∈ℤ}
  from A1 I have III: g∈𝒮 and IV: f∼g using Int_ZF_2_1_L33
    by auto
  from IV have ⟨f,g⟩ ∈ AlEqRel by simp
  with A1 have T: g ∈ 𝒮₊ by (rule Int_ZF_2_3_L9)
  moreover have ∀m∈ℤ₊. g⁻¹(m)-1 ∈ ℤ₊
  proof
    fix m assume A2: m∈ℤ₊
    from A1 I II have V: 2 ≤ g⁻¹(1)
      using Int_ZF_2_1_L33 PositiveSet_def by simp
    moreover from A2 T have g⁻¹(1) ≤ g⁻¹(m)
      using Int_ZF_1_5_L3 int_one_two_are_pos Int_ZF_2_4_L5
```

```
      by simp
    ultimately have 2 ≤ g⁻¹(m)
      by (rule Int_order_transitive)
    then have 2-1 ≤ g⁻¹(m)-1
      using int_zero_one_are_int Int_ZF_1_1_L4 int_ord_transl_inv
      by simp
    then show  g⁻¹(m)-1 ∈ ℤ₊
      using int_zero_one_are_int Int_ZF_1_2_L3 Int_ZF_1_5_L3
      by simp
  qed
  ultimately have ∃h∈𝒮. g∘h ∼ id(ℤ)
    by (rule Int_ZF_2_4_L12)
  with III IV show thesis by auto
qed
```

26.5 Completeness

In this section we consider properties of slopes that are needed for the proof of completeness of real numbers constructred in `Real_ZF_1.thy`. In particular we consider properties of embedding of integers into the set of slopes by the mapping $m \mapsto m^S$, where m^S is defined by $m^S(n) = m \cdot n$.

If m is an integer, then m^S is a slope whose value is $m \cdot n$ for every integer.

```
lemma (in int1) Int_ZF_2_5_L1: assumes A1: m ∈ ℤ
  shows
  ∀n ∈ ℤ. (m^S)(n) = m·n
  m^S ∈ 𝒮
proof -
  from A1 have I: m^S:ℤ→ℤ
    using Int_ZF_1_1_L5 ZF_fun_from_total by simp
  then show II: ∀n ∈ ℤ. (m^S)(n) = m·n using ZF_fun_from_tot_val
    by simp
  { fix n k
    assume A2: n∈ℤ  k∈ℤ
    with A1 have T: m·n ∈ ℤ  m·k ∈ ℤ
      using Int_ZF_1_1_L5 by auto
    from A1 A2 II T  have δ(m^S,n,k) = m·k - m·k
      using Int_ZF_1_1_L5 Int_ZF_1_1_L1 Int_ZF_1_2_L3
      by simp
    also from T have ... = 0 using Int_ZF_1_1_L4
      by simp
    finally have δ(m^S,n,k) = 0 by simp
    then have abs(δ(m^S,n,k)) ≤ 0
      using Int_ZF_2_L18 int_zero_one_are_int int_ord_is_refl refl_def
      by simp
  } then have ∀n∈ℤ.∀k∈ℤ. abs(δ(m^S,n,k)) ≤ 0
    by simp
  with I show  m^S ∈ 𝒮 by (rule Int_ZF_2_1_L5)
qed
```

For any slope f there is an integer m such that there is some slope g that is almost equal to m^S and dominates f in the sense that $f \leq g$ on positive integers (which implies that either g is almost equal to f or $g-f$ is a positive slope. This will be used in Real_ZF_1.thy to show that for any real number there is an integer that (whose real embedding) is greater or equal.

```
lemma (in int1) Int_ZF_2_5_L2: assumes A1: f ∈ 𝒮
  shows ∃m∈ℤ. ∃g∈𝒮. (m^S∼g ∧ (f∼g ∨ g+(-f) ∈ 𝒮₊))
proof -
  from A1 have
    ∃m k. m∈ℤ ∧ k∈ℤ ∧ (∀p∈ℤ. abs(f(p)) ≤ m·abs(p)+k)
    using Arthan_Lem_8 by simp
  then obtain m k where I: m∈ℤ and II: k∈ℤ and
    III: ∀p∈ℤ. abs(f(p)) ≤ m·abs(p)+k
    by auto
  let g = {⟨n,m^S(n) +k⟩. n∈ℤ}
  from I have IV: m^S ∈ 𝒮 using Int_ZF_2_5_L1 by simp
  with II have V: g∈𝒮 and VI: m^S∼g using Int_ZF_2_1_L33
    by auto
  { fix n assume A2: n∈ℤ₊
    with A1 have f(n) ∈ ℤ
      using Int_ZF_2_1_L2B PositiveSet_def by simp
    then have f(n) ≤ abs(f(n)) using Int_ZF_2_L19C
      by simp
    moreover
    from III A2 have abs(f(n)) ≤ m·abs(n) + k
      using PositiveSet_def by simp
    with A2 have abs(f(n)) ≤ m·n+k
      using Int_ZF_1_5_L4A by simp
    ultimately have f(n) ≤ m·n+k
      by (rule Int_order_transitive)
    moreover
    from II IV A2 have g(n) = (m^S)(n)+k
      using Int_ZF_2_1_L33 PositiveSet_def by simp
    with I A2 have g(n) = m·n+k
      using Int_ZF_2_5_L1 PositiveSet_def by simp
    ultimately have f(n) ≤ g(n)
      by simp
  } then have ∀n∈ℤ₊. f(n) ≤ g(n)
    by simp
  with A1 V have f∼g ∨ g + (-f) ∈ 𝒮₊
    using Int_ZF_2_3_L4C by simp
  with I V VI show thesis by auto
qed
```

The negative of an integer embeds in slopes as a negative of the orgiginal embedding.

```
lemma (in int1) Int_ZF_2_5_L3: assumes A1:  m ∈ ℤ
  shows (-m)^S = -(m^S)
```

```
proof -
  from A1 have (-m)^S: ℤ→ℤ and (-(m^S)): ℤ→ℤ
    using Int_ZF_1_1_L4 Int_ZF_2_5_L1 AlmostHoms_def Int_ZF_2_1_L12
    by auto
  moreover have ∀n∈ℤ. ((-m)^S)(n) = (-(m^S))(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 have
      ((-m)^S)(n) = (-m)·n
      (-(m^S))(n) = -(m·n)
      using Int_ZF_1_1_L4 Int_ZF_2_5_L1 Int_ZF_2_1_L12A
      by auto
    with A1 A2 show ((-m)^S)(n) = (-(m^S))(n)
      using Int_ZF_1_1_L5 by simp
  qed
  ultimately show (-m)^S = -(m^S) using fun_extension_iff
    by simp
qed
```

The sum of embeddings is the embeding of the sum.

```
lemma (in int1) Int_ZF_2_5_L3A: assumes A1: m∈ℤ  k∈ℤ
  shows (m^S) + (k^S) = ((m+k)^S)
proof -
  from A1 have T1: m+k ∈ ℤ using Int_ZF_1_1_L5
    by simp
  with A1 have T2:
    (m^S) ∈ 𝒮  (k^S) ∈ 𝒮
    (m+k)^S  ∈ 𝒮
    (m^S) + (k^S) ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L12C by auto
  then have
    (m^S) + (k^S) : ℤ→ℤ
    (m+k)^S : ℤ→ℤ
    using AlmostHoms_def by auto
  moreover have ∀n∈ℤ. ((m^S) + (k^S))(n) = ((m+k)^S)(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 T1 T2 have  ((m^S) + (k^S))(n) = (m+k)·n
      using Int_ZF_2_1_L12B Int_ZF_2_5_L1 Int_ZF_1_1_L1
      by simp
    also from T1 A2 have ... = ((m+k)^S)(n)
      using Int_ZF_2_5_L1 by simp
    finally show ((m^S) + (k^S))(n) = ((m+k)^S)(n)
      by simp
  qed
  ultimately show (m^S) + (k^S) = ((m+k)^S)
    using fun_extension_iff by simp
qed
```

The composition of embeddings is the embeding of the product.

```
lemma (in int1) Int_ZF_2_5_L3B: assumes A1: m∈ℤ  k∈ℤ
  shows (m^S) ∘ (k^S) = ((m·k)^S)
proof -
  from A1 have T1: m·k ∈ ℤ using Int_ZF_1_1_L5
    by simp
  with A1 have T2:
    (m^S) ∈ 𝒮  (k^S) ∈ 𝒮
    (m·k)^S  ∈ 𝒮
    (m^S) ∘ (k^S) ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L11 by auto
  then have
    (m^S) ∘ (k^S) : ℤ→ℤ
    (m·k)^S : ℤ→ℤ
    using AlmostHoms_def by auto
  moreover have ∀n∈ℤ. ((m^S) ∘ (k^S))(n) = ((m·k)^S)(n)
  proof
    fix n assume A2: n∈ℤ
    with A1 T2 have
      ((m^S) ∘ (k^S))(n) = (m^S)(k·n)
       using Int_ZF_2_1_L10 Int_ZF_2_5_L1 by simp
    moreover
    from A1 A2 have k·n ∈ ℤ using Int_ZF_1_1_L5
      by simp
    with A1 A2 have (m^S)(k·n) = m·k·n
      using Int_ZF_2_5_L1 Int_ZF_1_1_L7 by simp
    ultimately have ((m^S) ∘ (k^S))(n) = m·k·n
      by simp
    also from T1 A2 have m·k·n = ((m·k)^S)(n)
      using Int_ZF_2_5_L1 by simp
    finally show ((m^S) ∘ (k^S))(n) = ((m·k)^S)(n)
      by simp
  qed
  ultimately show (m^S) ∘ (k^S) = ((m·k)^S)
    using fun_extension_iff by simp
qed
```

Embedding integers in slopes preserves order.

```
lemma (in int1) Int_ZF_2_5_L4: assumes A1:  m≤n
  shows (m^S) ∼ (n^S) ∨ (n^S)+(-(m^S)) ∈ 𝒮₊
proof -
  from A1 have m^S ∈ 𝒮 and n^S ∈ 𝒮
    using Int_ZF_2_L1A Int_ZF_2_5_L1 by auto
  moreover from A1 have ∀k∈ℤ₊. (m^S)(k) ≤ (n^S)(k)
    using Int_ZF_1_3_L13B Int_ZF_2_L1A PositiveSet_def Int_ZF_2_5_L1
    by simp
  ultimately show thesis using Int_ZF_2_3_L4C
    by simp
qed
```

We aim at showing that $m \mapsto m^S$ is an injection modulo the relation of almost equality. To do that we first show that if m^S has finite range, then $m = 0$.

```
lemma (in int1) Int_ZF_2_5_L5:
  assumes m∈ℤ and m^S ∈ FinRangeFunctions(ℤ,ℤ)
  shows m=0
  using prems FinRangeFunctions_def Int_ZF_2_5_L1 AlmostHoms_def
    func_imagedef Int_ZF_1_6_L8 by simp
```

Embeddings of two integers are almost equal only if the integers are equal.

```
lemma (in int1) Int_ZF_2_5_L6:
  assumes A1: m∈ℤ  k∈ℤ and A2: (m^S) ∼ (k^S)
  shows m=k
proof -
  from A1 have T: m-k ∈ ℤ using Int_ZF_1_1_L5 by simp
  from A1 have (-(k^S)) =  ((-k)^S)
    using Int_ZF_2_5_L3 by simp
  then have m^S + (-(k^S)) = (m^S) + ((-k)^S)
    by simp
  with A1 have m^S + (-(k^S)) = ((m-k)^S)
    using Int_ZF_1_1_L4 Int_ZF_2_5_L3A by simp
  moreover from A1 A2 have m^S + (-(k^S)) ∈ FinRangeFunctions(ℤ,ℤ)
    using Int_ZF_2_5_L1 Int_ZF_2_1_L9D by simp
  ultimately have (m-k)^S ∈ FinRangeFunctions(ℤ,ℤ)
    by simp
  with T have m-k = 0 using Int_ZF_2_5_L5
    by simp
  with A1 show m=k by (rule Int_ZF_1_L15)
qed
```

Embedding of 1 is the identity slope and embedding of zero is a finite range function.

```
lemma (in int1) Int_ZF_2_5_L7: shows
  1^S = id(ℤ)
  0^S ∈ FinRangeFunctions(ℤ,ℤ)
proof -
  have id(ℤ) = {⟨x,x⟩. x∈ℤ}
    using id_def by blast
  then show 1^S = id(ℤ) using Int_ZF_1_1_L4 by simp
  have {0^S(n). n∈ℤ} = {0·n. n∈ℤ}
    using int_zero_one_are_int Int_ZF_2_5_L1 by simp
  also have ... = {0} using Int_ZF_1_1_L4 int_not_empty
    by simp
  finally have {0^S(n). n∈ℤ} = {0} by simp
  then have {0^S(n). n∈ℤ} ∈ Fin(ℤ)
    using int_zero_one_are_int Finite1_L16 by simp
  moreover have 0^S: ℤ→ℤ
    using int_zero_one_are_int Int_ZF_2_5_L1 AlmostHoms_def
    by simp
```

```
  ultimately show 0^S ∈ FinRangeFunctions(ℤ,ℤ)
    using Finite1_L19 by simp
qed
```

A somewhat technical condition for a embedding of an integer to be "less or equal" (in the sense apriopriate for slopes) than the composition of a slope and another integer (embedding).

```
lemma (in int1) Int_ZF_2_5_L8:
  assumes A1: f ∈ 𝒮 and A2: N ∈ ℤ  M ∈ ℤ and
  A3: ∀n∈ℤ+. M·n ≤ f(N·n)
  shows M^S ∼ f∘(N^S) ∨  (f∘(N^S)) + (-(M^S)) ∈ 𝒮+
proof -
  from A1 A2 have M^S ∈ 𝒮  f∘(N^S) ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L11 by auto
  moreover from A1 A2 A3 have ∀n∈ℤ+. (M^S)(n) ≤ (f∘(N^S))(n)
    using Int_ZF_2_5_L1 PositiveSet_def Int_ZF_2_1_L10
    by simp
  ultimately show thesis using Int_ZF_2_3_L4C
    by simp
qed
```

Another technical condition for the composition of a slope and an integer (embedding) to be "less or equal" (in the sense apriopriate for slopes) than embedding of another integer.

```
lemma (in int1) Int_ZF_2_5_L9:
  assumes A1: f ∈ 𝒮 and A2: N ∈ ℤ  M ∈ ℤ and
  A3: ∀n∈ℤ+.  f(N·n) ≤ M·n
  shows f∘(N^S) ∼ (M^S) ∨ (M^S) + (-(f∘(N^S))) ∈ 𝒮+
proof -
  from A1 A2 have f∘(N^S) ∈ 𝒮  M^S ∈ 𝒮
    using Int_ZF_2_5_L1 Int_ZF_2_1_L11 by auto
  moreover from A1 A2 A3 have ∀n∈ℤ+. (f∘(N^S))(n) ≤ (M^S)(n)
    using Int_ZF_2_5_L1 PositiveSet_def Int_ZF_2_1_L10
    by simp
  ultimately show thesis using Int_ZF_2_3_L4C
    by simp
qed



end
```

27 Real_ZF.thy

theory `Real_ZF` **imports** `Int_ZF 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 $s : Z \rightarrow Z$ 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.

27.1 The definition of real numbers

First we define slopes and real numbers as the set of their classes. The definition of slopes references the notion of almost homomorphisms defined in `Group_ZF_2.thy`: slopes are defined as almost homomorphisms on integers with integer addition as the operation. Similarly the notions of the first and second operation on slopes (which is really the addition and composition of slopes) is derived as a special case of the first and second operation on almost homomorphisms.

constdefs

```
Slopes ≡ AlmostHoms(int,IntegerAddition)

SlopeOp1 ≡ AlHomOp1(int,IntegerAddition)

SlopeOp2 ≡ AlHomOp2(int,IntegerAddition)

BoundedIntMaps ≡ FinRangeFunctions(int,int)

SlopeEquivalenceRel ≡ QuotientGroupRel(Slopes,SlopeOp1,BoundedIntMaps)

RealNumbers ≡ Slopes//SlopeEquivalenceRel

RealAddition ≡ ProjFun2(Slopes,SlopeEquivalenceRel,SlopeOp1)
```

```
  RealMultiplication ≡ ProjFun2(Slopes,SlopeEquivalenceRel,SlopeOp2)
```

We first show that we can use theorems proven in some proof contexts (locales). The locale group1 requires assumption that we deal with an abelian group. The next lemma allows to use all theorems proven in the context called group1.

```
lemma Real_ZF_1_L1: shows group1(int,IntegerAddition)
  using group1_axioms.intro group1_def Int_ZF_1_T2 by simp
```

Real numbers form a ring. This is a special case of the theorem proven in Ring_ZF_1.thy, where we show the same in general for almost homomorphisms rather than slopes.

```
theorem Real_ZF_1_T1: IsAring(RealNumbers,RealAddition,RealMultiplication)
proof -
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let Op2 = AlHomOp2(int,IntegerAddition)
  let R = QuotientGroupRel(AH,Op1,FR)
  let A = ProjFun2(AH,R,Op1)
  let M = ProjFun2(AH,R,Op2)
  have IsAring(AH//R,A,M) using Real_ZF_1_L1 group1.Ring_ZF_1_1_T1
    by simp
  then show thesis using Slopes_def SlopeOp2_def SlopeOp1_def
    BoundedIntMaps_def SlopeEquivalenceRel_def RealNumbers_def
    RealAddition_def RealMultiplication_def by simp
qed
```

We can use theorems proven in group0 and group1 contexts applied to the group of real numbers.

```
lemma Real_ZF_1_L2:
  group0(RealNumbers,RealAddition)
  RealAddition {is commutative on} RealNumbers
  group1(RealNumbers,RealAddition)
proof -
  have
    IsAgroup(RealNumbers,RealAddition)
    RealAddition {is commutative on} RealNumbers
    using Real_ZF_1_T1 IsAring_def by auto
  then show
    group0(RealNumbers,RealAddition)
    RealAddition {is commutative on} RealNumbers
    group1(RealNumbers,RealAddition)
    using group1_axioms.intro group0_def group1_def
    by auto
qed
```

Let's define some notation.

locale real0 =

```
  fixes real (ℝ)
  defines real_def [simp]: ℝ ≡ RealNumbers

  fixes ra (infixl + 69)
  defines ra_def [simp]: a+ b ≡ RealAddition⟨a,b⟩

  fixes rminus :: i⇒i (- _ 72)
  defines rminus_def [simp]:-a ≡ GroupInv(ℝ,RealAddition)(a)

  fixes rsub (infixl - 69)
  defines rsub_def [simp]: a-b ≡ a+(-b)

  fixes rm (infixl · 70)
  defines rm_def [simp]: a·b ≡ RealMultiplication⟨a,b⟩

  fixes rzero (0)
  defines rzero_def [simp]:
  0 ≡ TheNeutralElement(RealNumbers,RealAddition)

  fixes rone (1)
  defines rone_def [simp]:
  1 ≡ TheNeutralElement(RealNumbers,RealMultiplication)

  fixes rtwo (2)
  defines rtwo_def [simp]: 2  ≡ 1+1

  fixes non_zero (ℝ0)
  defines non_zero_def[simp]: ℝ0 ≡ ℝ-{0}

  fixes inv (_^-1  [90] 91)
  defines inv_def[simp]:
  a^-1 ≡ GroupInv(ℝ0,restrict(RealMultiplication,ℝ0×ℝ0))(a)
```

In real0 context all theorems proven in the ring0, context are valid.

```
lemma (in real0) Real_ZF_1_L3: shows
  ring0(ℝ,RealAddition,RealMultiplication)
  using Real_ZF_1_T1 ring0_def ring0.Ring_ZF_1_L1
  by auto
```

Lets try out our notation to see that zero and one are real numbers.

```
lemma (in real0) Real_ZF_1_L4: shows 0∈ℝ  1∈ℝ
  using Real_ZF_1_L3 ring0.Ring_ZF_1_L2 by auto
```

The lemma below lists some properties that require one real number to state.

```
lemma (in real0) Real_ZF_1_L5: assumes A1: a∈ℝ
  shows
  (-a) ∈ ℝ
```

```
(-(-a)) = a
a+0 = a
0+a = a
a·1 = a
1·a = a
a-a = 0
a-0 = a
using prems Real_ZF_1_L3 ring0.Ring_ZF_1_L3 by auto
```

The lemma below lists some properties that require two real numbers to state.

```
lemma (in real0) Real_ZF_1_L6: assumes a∈ℝ  b∈ℝ
  shows
  a+b ∈ ℝ
  a-b ∈ ℝ
  a·b ∈ ℝ
  a+b = b+a
  (-a)·b = -(a·b)
  a·(-b) = -(a·b)
  using prems Real_ZF_1_L3 ring0.Ring_ZF_1_L4 ring0.Ring_ZF_1_L7
  by auto
```

Multiplication of reals is associative.

```
lemma (in real0) Real_ZF_1_L6A: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows a·(b·c) = (a·b)·c
  using prems Real_ZF_1_L3 ring0.Ring_ZF_1_L11
  by simp
```

Addition is distributive with respect to multiplication.

```
lemma (in real0) Real_ZF_1_L7: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows
  a·(b+c) = a·b + a·c
  (b+c)·a = b·a + c·a
  a·(b-c) = a·b - a·c
  (b-c)·a = b·a - c·a
  using prems Real_ZF_1_L3 ring0.ring_oper_distr  ring0.Ring_ZF_1_L8
  by auto
```

A simple rearrangement with four real numbers.

```
lemma (in real0) Real_ZF_1_L7A:
  assumes a∈ℝ  b∈ℝ  c∈ℝ  d∈ℝ
  shows a-b + (c-d) = a+c-b-d
  using prems Real_ZF_1_L2 group0.group0_4_L8A by simp
```

`RealAddition` is defined as the projection of the first operation on slopes (that is, slope addition) on the quotient (slopes divided by the "almost equal" relation. The next lemma plays with definitions to show that this is the same as the operation induced on the appriopriate quotient group.

The names AH, Op1 and FR are used in group1 context to denote almost homomorphisms, the first operation on AH and finite range functions resp.

```
lemma Real_ZF_1_L8: assumes
  AH = AlmostHoms(int,IntegerAddition) and
  Op1 = AlHomOp1(int,IntegerAddition) and
  FR = FinRangeFunctions(int,int)
  shows RealAddition = QuotientGroupOp(AH,Op1,FR)
  using prems RealAddition_def SlopeEquivalenceRel_def
    QuotientGroupOp_def Slopes_def SlopeOp1_def BoundedIntMaps_def
  by simp
```

The symbol **0** in the real0 context is defined as the neutral element of real addition. The next lemma shows that this is the same as the neutral element of the appriopriate quotient group.

```
lemma (in real0) Real_ZF_1_L9: assumes
  AH = AlmostHoms(int,IntegerAddition) and
  Op1 = AlHomOp1(int,IntegerAddition) and
  FR = FinRangeFunctions(int,int) and
  r = QuotientGroupRel(AH,Op1,FR)
  shows
  TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR)) = 0
  SlopeEquivalenceRel = r
  using prems Slopes_def Real_ZF_1_L8 RealNumbers_def
    SlopeEquivalenceRel_def SlopeOp1_def BoundedIntMaps_def
  by auto
```

Zero is the class of any finite range function.

```
lemma (in real0) Real_ZF_1_L10:
  assumes A1: s ∈ Slopes
  shows SlopeEquivalenceRel{s} = 0 ←→ s ∈ BoundedIntMaps
proof -
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let r = QuotientGroupRel(AH,Op1,FR)
  let e = TheNeutralElement(AH//r,QuotientGroupOp(AH,Op1,FR))
  from A1 have
    group1(int,IntegerAddition)
    s∈AH
    using Real_ZF_1_L1 Slopes_def
    by auto
  then have r{s} = e ←→ s ∈ FR
    using group1.Group_ZF_3_3_L5 by simp
  moreover have
    r = SlopeEquivalenceRel
    e = 0
    FR = BoundedIntMaps
    using SlopeEquivalenceRel_def Slopes_def SlopeOp1_def
```

```
      BoundedIntMaps_def Real_ZF_1_L9 by auto
  ultimately show thesis by simp
qed
```

We will need a couple of results from Group_ZF_3.thy The first two that state that the definition of addition and multiplication of real numbers are consistent, that is the result does not depend on the choice of the slopes representing the numbers. The second one implies that what we call SlopeEquivalenceRel is actually an equivalence relation on the set of slopes. We also show that the neutral element of the multiplicative operation on reals (in short number 1) is the class of the identity function on integers.

```
lemma Real_ZF_1_L11: shows
  Congruent2(SlopeEquivalenceRel,SlopeOp1)
  Congruent2(SlopeEquivalenceRel,SlopeOp2)
  SlopeEquivalenceRel ⊆ Slopes × Slopes
  equiv(Slopes, SlopeEquivalenceRel)
  SlopeEquivalenceRel{id(int)} =
  TheNeutralElement(RealNumbers,RealMultiplication)
  BoundedIntMaps ⊆ Slopes
proof -
  let G = int
  let f = IntegerAddition
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let Op2 = AlHomOp2(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let R = QuotientGroupRel(AH,Op1,FR)
   have
     Congruent2(R,Op1)
     Congruent2(R,Op2)
    using Real_ZF_1_L1 group1.Group_ZF_3_4_L13A group1.Group_ZF_3_3_L4
    by auto
  then show
    Congruent2(SlopeEquivalenceRel,SlopeOp1)
    Congruent2(SlopeEquivalenceRel,SlopeOp2)
    using SlopeEquivalenceRel_def SlopeOp1_def Slopes_def
      BoundedIntMaps_def SlopeOp2_def by auto
  have equiv(AH,R)
    using Real_ZF_1_L1 group1.Group_ZF_3_3_L3 by simp
  then show equiv(Slopes,SlopeEquivalenceRel)
    using BoundedIntMaps_def SlopeEquivalenceRel_def SlopeOp1_def Slopes_def
    by simp
  then show SlopeEquivalenceRel ⊆ Slopes × Slopes
    using equiv_type by simp
  have R{id(int)} = TheNeutralElement(AH//R,ProjFun2(AH,R,Op2))
    using Real_ZF_1_L1 group1.Group_ZF_3_4_T2 by simp
  then show  SlopeEquivalenceRel{id(int)} =
    TheNeutralElement(RealNumbers,RealMultiplication)
```

```
    using Slopes_def RealNumbers_def
    SlopeEquivalenceRel_def SlopeOp1_def BoundedIntMaps_def
    RealMultiplication_def SlopeOp2_def
    by simp
  have FR ⊆ AH using Real_ZF_1_L1 group1.Group_ZF_3_3_L1
    by simp
  then show BoundedIntMaps ⊆ Slopes
    using BoundedIntMaps_def Slopes_def by simp
qed
```

A one-side implication of the equivalence from Real_ZF_1_L10: the class of a bounded integer map is the real zero.

```
lemma (in real0) Real_ZF_1_L11A: assumes s ∈ BoundedIntMaps
  shows SlopeEquivalenceRel{s} = 0
  using prems Real_ZF_1_L11 Real_ZF_1_L10 by auto
```

The next lemma is rephrases the result from Group_ZF_3.thy that says that the negative (the group inverse with respect to real addition) of the class of a slope is the class of that slope composed with the integer additive group inverse. The result and proof is not very readable as we use mostly generic set theory notation with long names here. Real_ZF_1.thy contains the same statement written in a more readable notation: $[-s] = -[s]$.

```
lemma (in real0) Real_ZF_1_L12: assumes A1: s ∈ Slopes and
  Dr: r = QuotientGroupRel(Slopes,SlopeOp1,BoundedIntMaps)
  shows r{GroupInv(int,IntegerAddition) O s} = -(r{s})
proof -
  let G = int
  let f = IntegerAddition
  let AH = AlmostHoms(int,IntegerAddition)
  let Op1 = AlHomOp1(int,IntegerAddition)
  let FR = FinRangeFunctions(int,int)
  let F = ProjFun2(Slopes,r,SlopeOp1)
  from A1 Dr have
    group1(G, f)
    s ∈ AlmostHoms(G, f)
    r = QuotientGroupRel(
    AlmostHoms(G, f), AlHomOp1(G, f), FinRangeFunctions(G, G))
    and F = ProjFun2(AlmostHoms(G, f), r, AlHomOp1(G, f))
    using Real_ZF_1_L1 Slopes_def SlopeOp1_def BoundedIntMaps_def
    by auto
  then have
    r{GroupInv(G, f) O s} =
    GroupInv(AlmostHoms(G, f) // r, F)(r  {s})
    using group1.Group_ZF_3_3_L6 by simp
  with Dr show thesis
    using RealNumbers_def Slopes_def SlopeEquivalenceRel_def RealAddition_def
    by simp
qed
```

Two classes are equal iff the slopes that represent them are almost equal.

```
lemma Real_ZF_1_L13: assumes s ∈ Slopes  p ∈ Slopes
  and r = SlopeEquivalenceRel
  shows r{s} = r{p} ⟷ ⟨s,p⟩ ∈  r
  using prems Real_ZF_1_L11 eq_equiv_class equiv_class_eq
  by blast
```

Identity function on integers is a slope.

```
lemma Real_ZF_1_L14: shows id(int) ∈ Slopes
proof -
  have id(int) ∈ AlmostHoms(int,IntegerAddition)
    using Real_ZF_1_L1 group1.Group_ZF_3_4_L15
    by simp
  then show thesis using Slopes_def by simp
qed
```

This 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.

```
end
```

28 Real_ZF_1.thy

theory `Real_ZF_1` **imports** `Real_ZF Int_ZF_2 OrderedField_ZF`

begin

In this theory file we continue the construction of real numbers started in `Real_ZF.thy` 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.

28.1 Definitions and notation

In this section we define notions and notation needed for the rest of the construction.

The order on the set of real numbers is constructed by specifying the set of positive reals. This is defined as the projection of the set of positive slopes. A slope is positive if it takes an infinite number of posititive values on the positive integers (see `Int_ZF_2.thy` for properties of positive slopes). The order relation on real numbers is defined by prescribing the set of positive numbers (see section "Alternative definitions" in `OrderedGroup_ZF.thy.`).

constdefs

```
PositiveSlopes ≡ {s ∈ Slopes.
s(PositiveIntegers) ∩  PositiveIntegers ∉ Fin(int)}

PositiveReals ≡ {SlopeEquivalenceRel{s}. s ∈ PositiveSlopes}

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 (∘ 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] ≡ SlopeEquivalenceRel{f}


fixes slope_neg :: i⇒i (-_ [90] 91)
defines slope_neg_def [simp]: -s ≡ GroupInv(int,IntegerAddition) O s

fixes lesseqr (infix ≤ 60)
defines lesseqr_def [simp]: a ≤ b ≡ ⟨a,b⟩ ∈ OrderOnReals

fixes sless (infix < 60)
defines sless_def [simp]: a < b ≡ a≤b ∧ a≠b

fixes positivereals (ℝ₊)
defines positivereals_def [simp]: ℝ₊ ≡ PositiveSet(ℝ,RealAddition,OrderOnReals)

fixes intembed (_^R [90] 91)
defines intembed_def [simp]:
m^R ≡ [{⟨n,IntegerMultiplication⟨m,n⟩ ⟩. n ∈ int}]

fixes floor (⌊ _ ⌋)
defines floor_def [simp]:
⌊a⌋ ≡ Maximum(IntegerOrder,{m ∈ int. m^R ≤ a})

fixes Γ
defines Γ_def [simp]: Γ(S,p) ≡ Maximum(IntegerOrder,{⌊p^R·x⌋. x∈S})

fixes ia (infixl + 69)
defines ia_def [simp]: a+b ≡ IntegerAddition<a,b>

fixes iminus :: i⇒i (- _ 72)
defines rminus_def [simp]: -a ≡ GroupInv(int,IntegerAddition)(a)

fixes isub (infixl - 69)
defines isub_def [simp]: a-b ≡ a+ (- b)
```

```
  fixes intpositives (ℤ+)
  defines intpositives_def [simp]:
  ℤ+ ≡ PositiveSet(int,IntegerAddition,IntegerOrder)

  fixes zlesseq (infix ≤ 60)
  defines lesseq_def [simp]: m ≤ n ≡ ⟨m,n⟩ ∈ IntegerOrder

  fixes imult (infixl · 70)
  defines imult_def [simp]: a·b ≡ IntegerMultiplication<a,b>

  fixes izero (0_Z)
  defines izero_def [simp]: 0_Z ≡ 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)
```

28.2 Multiplication of real numbers

Multiplication of real numbers is defined as a projection of composition of slopes onto the space of equivalence classes of slopes. Thus, the product of the real numbers given as classes of slopes s and r is defined as the class of $s \circ r$. The goal of this section is to show that multiplication defined this way is commutative.

Let's recall a theorem from `Int_ZF_2.thy` that states that if f, g are slopes, then $f \circ g$ is equivalent to $g \circ f$. Here we conclude from that that the classes of $f \circ g$ and $g \circ f$ are the same.

```
lemma (in real1) Real_ZF_1_1_L2: assumes A1: f ∈ S  g ∈ S
  shows [f∘g] = [g∘f]
proof -
  from A1 have f∘g ∼ g∘f
    using Slopes_def int1.Arthan_Th_9 SlopeOp1_def BoundedIntMaps_def
      SlopeEquivalenceRel_def SlopeOp2_def by simp
  then show thesis using  Real_ZF_1_L11 equiv_class_eq
    by simp
qed
```

Classes of slopes are real numbers.

```
lemma (in real1) Real_ZF_1_1_L3: assumes A1: f ∈ 𝒮
  shows [f] ∈ ℝ
proof -
  from A1 have [f] ∈ Slopes//SlopeEquivalenceRel
    using Slopes_def quotientI by simp
  then show [f] ∈ ℝ using RealNumbers_def by simp
qed
```

Each real number is a class of a slope.

```
lemma (in real1) Real_ZF_1_1_L3A: assumes A1: a∈ℝ
  shows ∃f∈𝒮 . a = [f]
proof -
  from A1 have a ∈ 𝒮//SlopeEquivalenceRel
    using RealNumbers_def Slopes_def by simp
  then show thesis using quotient_def
    by simp
qed
```

It is useful to have the definition of addition and multiplication in the `real1` context notation.

```
lemma (in real1) Real_ZF_1_1_L4:
  assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows
  [f] + [g] = [f+g]
  [f] · [g] = [f∘g]
proof -
  let r = SlopeEquivalenceRel
  have [f]·[g] = ProjFun2(𝒮,r,SlopeOp2)⟨[f],[g]⟩
    using RealMultiplication_def Slopes_def by simp
  also from A1 have ... = [f∘g]
    using Real_ZF_1_L11 EquivClass_1_L10 Slopes_def
    by simp
  finally show [f] · [g] = [f∘g] by simp
  have [f] + [g] = ProjFun2(𝒮,r,SlopeOp1)⟨[f],[g]⟩
    using RealAddition_def Slopes_def by simp
  also from A1 have ... = [f+g]
    using Real_ZF_1_L11 EquivClass_1_L10 Slopes_def
    by simp
  finally show [f] + [g] = [f+g] by simp
qed
```

The next lemma is essentially the same as `Real_ZF_1_L12`, but written in the notation defined in the `real1` context. It states that if f is a slope, then $-[f] = [-f]$.

```
lemma (in real1) Real_ZF_1_1_L4A: assumes f ∈ 𝒮
  shows [-f] = -[f]
  using prems Slopes_def SlopeEquivalenceRel_def Real_ZF_1_L12
  by simp
```

Subtracting real numbers correspods to adding the opposite slope.

```
lemma (in real1) Real_ZF_1_1_L4B: assumes A1: f ∈ 𝒮  g ∈ 𝒮
  shows [f] - [g] = [f+(-g)]
proof -
  from A1 have [f+(-g)] = [f] + [-g]
    using Slopes_def BoundedIntMaps_def int1.Int_ZF_2_1_L12
      Real_ZF_1_1_L4 by simp
  with A1 show [f] - [g] = [f+(-g)]
    using Real_ZF_1_1_L4A by simp
qed
```

Multiplication of real numbers is commutative.

```
theorem (in real1) real_mult_commute: assumes A1: a∈ℝ  b∈ℝ
  shows a·b = b·a
proof -
  from A1 have
    ∃f∈𝒮 . a = [f]
    ∃g∈𝒮 . b = [g]
    using Real_ZF_1_1_L3A by auto
  then obtain f g where
    f ∈ 𝒮  g ∈ 𝒮 and a = [f]  b = [g]
    by auto
  then show a·b = b·a
    using Real_ZF_1_1_L4 Real_ZF_1_1_L2 by simp
qed
```

Multiplication is commutative on reals.

```
lemma real_mult_commutative: shows
  RealMultiplication {is commutative on} RealNumbers
  using real1.real_mult_commute IsCommutative_def
  by simp
```

The neutral element of multiplication of reals (denoted as **1** in the `real1` context) is the class of identity function on integers. This is really shown in `Real_ZF_1_L11`, here we only rewrite it in the notation used in the `real1` context.

```
lemma (in real1) real_one_cl_identity: shows [id(int)] = 1
  using Real_ZF_1_L11 by simp
```

If f is bounded, then its class is the neutral element of additive operation on reals (denoted as **0** in the `real1` context).

```
lemma (in real1) real_zero_cl_bounded_map:
  assumes f ∈ BoundedIntMaps shows [f] = 0
  using prems Real_ZF_1_L11A by simp
```

Two real numbers are equal iff the slopes that represent them are almost equal. This is proven in `Real_ZF_1_L13`, here we just rewrite it in the notation used in the `real1` context.

```
lemma (in real1) Real_ZF_1_1_L5:
  assumes f ∈ 𝒮  g ∈ 𝒮
  shows [f] = [g] ⟷ f ∼ g
  using prems Slopes_def Real_ZF_1_L13 by simp
```

If the pair of function belongs to the slope equivalence relation, then their classes are equal. This is convenient, because we don't need to assume that f, g are slopes (follows from the fact that $f \sim g$).

```
lemma (in real1) Real_ZF_1_1_L5A: assumes f ∼ g
  shows [f] = [g]
  using prems Real_ZF_1_L11 Slopes_def Real_ZF_1_1_L5
  by auto
```

Identity function on integers is a slope. This is proven in `Real_ZF_1_L13`, here we just rewrite it in the notation used in the `real1` context.

```
lemma (in real1) id_on_int_is_slope: shows id(int) ∈ 𝒮
  using Real_ZF_1_L14 Slopes_def by simp
```

A result from `Int_ZF_2.thy`: the identity function on integers is not almost equal to any bounded function.

```
lemma (in real1) Real_ZF_1_1_L7:
  assumes A1: f ∈ BoundedIntMaps
  shows ¬(id(int) ∼ f)
  using prems Slopes_def SlopeOp1_def BoundedIntMaps_def
    SlopeEquivalenceRel_def BoundedIntMaps_def int1.Int_ZF_2_3_L12
  by simp
```

Zero is not one.

```
lemma (in real1) real_zero_not_one: shows 1≠0
proof -
  { assume A1: 1=0
    have ∃f ∈ 𝒮. 0 = [f]
      using  Real_ZF_1_L4 Real_ZF_1_1_L3A by simp
    with A1 have
      ∃f ∈ 𝒮. [id(int)] = [f] ∧ [f] = 0
      using real_one_cl_identity by auto
    then have False using Real_ZF_1_1_L5 Slopes_def
      Real_ZF_1_L10 Real_ZF_1_1_L7 id_on_int_is_slope
      by auto
  } then show 1≠0 by auto
qed
```

Negative of a real number is a real number. Property of groups.

```
lemma (in real1) Real_ZF_1_1_L8: assumes a∈ℝ shows (-a) ∈ ℝ
  using prems Real_ZF_1_L2 group0.inverse_in_group
  by simp
```

An identity with three real numbers.

```
lemma (in real1) Real_ZF_1_1_L9: assumes a∈ℝ  b∈ℝ  c∈ℝ
  shows a·(b·c) = a·c·b
  using prems real_mult_commutative Real_ZF_1_L3 ring0.Ring_ZF_2_L4
  by simp
```

28.3 The order on reals

In this section we show that the order relation defined by prescribing the set of positive reals as the projection of the set of positive slopes makes the ring of real numbers into an ordered ring. We also collect the facts about ordered groups and rings that we use in the construction.

Positive slopes are slopes and positive reals are real.

```
lemma Real_ZF_1_2_L1: shows
  PositiveSlopes ⊆ Slopes
  PositiveReals ⊆ RealNumbers
proof -
  have PositiveSlopes =
    {s ∈ Slopes. s(PositiveIntegers) ∩ PositiveIntegers ∉ Fin(int)}
    using PositiveSlopes_def by simp
  then show PositiveSlopes ⊆ Slopes by (rule subset_with_property)
  then have
    {SlopeEquivalenceRel{s}. s ∈ PositiveSlopes } ⊆
    Slopes//SlopeEquivalenceRel
    using EquivClass_1_L1A by simp
  then show PositiveReals ⊆ RealNumbers
    using PositiveReals_def RealNumbers_def by simp
qed
```

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

```
lemma (in real1) Real_ZF_1_2_L2:
  shows a ∈ PositiveReals ⟷ (∃f∈𝒮₊. a = [f])
proof
  assume a ∈ PositiveReals
  then have a ∈ {([s]). s ∈ 𝒮₊} using PositiveReals_def
    by simp
  then show ∃f∈𝒮₊. a = [f] by auto
next assume ∃f∈𝒮₊. a = [f]
  then have  a ∈ {([s]). s ∈ 𝒮₊} by auto
  then show a ∈ PositiveReals using PositiveReals_def
    by simp
qed
```

Let's recall from `Int_ZF_2.thy` that the sum and composition of positive slopes is a positive slope.

```
lemma (in real1) Real_ZF_1_2_L3:
  assumes f∈𝒮₊  g∈𝒮₊
  shows
```

```
  f+g ∈ 𝒮₊
  f∘g ∈ 𝒮₊
  using prems Slopes_def PositiveSlopes_def PositiveIntegers_def
    SlopeOp1_def int1.sum_of_pos_sls_is_pos_sl
    SlopeOp2_def int1.comp_of_pos_sls_is_pos_sl
  by auto
```

Bounded integer maps are not positive slopes.

```
lemma (in real1) Real_ZF_1_2_L5:
  assumes f ∈ BoundedIntMaps
  shows f ∉ 𝒮₊
  using prems BoundedIntMaps_def Slopes_def PositiveSlopes_def
    PositiveIntegers_def int1.Int_ZF_2_3_L1B by simp
```

The set of positive reals is closed under addition and multiplication. Zero (the neutral element of addition) is not a positive number.

```
lemma (in real1) Real_ZF_1_2_L6: shows
  PositiveReals {is closed under} RealAddition
  PositiveReals {is closed under} RealMultiplication
  0 ∉ PositiveReals
proof -
  { fix a fix b
    assume a ∈ PositiveReals and b ∈ PositiveReals
    then obtain f g where
      I: f ∈ 𝒮₊  g ∈ 𝒮₊ and
      II: a = [f]  b = [g]
      using Real_ZF_1_2_L2 by auto
    then have f ∈ 𝒮  g ∈ 𝒮 using Real_ZF_1_2_L1 Slopes_def
      by auto
    with I II have
      a+b ∈ PositiveReals ∧ a·b ∈ PositiveReals
       using Real_ZF_1_1_L4 Real_ZF_1_2_L3 Real_ZF_1_2_L2
       by auto
  } then show
      PositiveReals {is closed under} RealAddition
      PositiveReals {is closed under} RealMultiplication
    using IsOpClosed_def
    by auto
  { assume 0 ∈ PositiveReals
    then obtain f where f ∈ 𝒮₊ and 0 = [f]
      using Real_ZF_1_2_L2 by auto
    then have False
      using Real_ZF_1_2_L1 Slopes_def Real_ZF_1_L10 Real_ZF_1_2_L5
      by auto
  } then show 0 ∉ PositiveReals by auto
qed
```

If a class of a slope f is not zero, then either f is a positive slope or $-f$ is a positive slope. The real proof is in `Int_ZF_2.thy`.

```
lemma (in real1) Real_ZF_1_2_L7:
  assumes A1: f ∈ 𝒮 and A2: [f] ≠ 0
  shows (f ∈ 𝒮₊) Xor ((-f) ∈ 𝒮₊)
  using prems Slopes_def SlopeEquivalenceRel_def BoundedIntMaps_def
    PositiveSlopes_def PositiveIntegers_def
    Real_ZF_1_L10 int1.Int_ZF_2_3_L8 by simp
```

The next lemma rephrases Int_ZF_2_3_L10 in the notation used in real1 context.

```
lemma (in real1) Real_ZF_1_2_L8:
  assumes A1: f ∈ 𝒮  g ∈ 𝒮
  and A2: (f ∈ 𝒮₊) Xor (g ∈ 𝒮₊)
  shows ([f] ∈ PositiveReals) Xor ([g] ∈ PositiveReals)
  using prems PositiveReals_def SlopeEquivalenceRel_def Slopes_def
    SlopeOp1_def BoundedIntMaps_def PositiveSlopes_def PositiveIntegers_def
    int1.Int_ZF_2_3_L10 by simp
```

The trichotomy law for the (potential) order on reals: if $a \neq 0$, then either a is positive or $-a$ is potitive.

```
lemma (in real1) Real_ZF_1_2_L9:
  assumes A1: a∈ℝ and A2: a≠0
  shows (a ∈ PositiveReals) Xor ((-a) ∈ PositiveReals)
proof -
  from A1 obtain f where I: f ∈ 𝒮  a = [f]
    using Real_ZF_1_1_L3A by auto
  with A2 have ([f] ∈ PositiveReals) Xor ([-f] ∈ PositiveReals)
    using Slopes_def BoundedIntMaps_def int1.Int_ZF_2_1_L12
      Real_ZF_1_2_L7 Real_ZF_1_2_L8 by simp
  with I show (a ∈ PositiveReals) Xor ((-a) ∈ PositiveReals)
    using Real_ZF_1_1_L4A by simp
qed
```

Finally we are ready to prove that real numbers form an ordered ring. with no zero divisors.

```
theorem reals_are_ord_ring: shows
  IsAnOrdRing(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  OrderOnReals {is total on} RealNumbers
  PositiveSet(RealNumbers,RealAddition,OrderOnReals) = PositiveReals
  HasNoZeroDivs(RealNumbers,RealAddition,RealMultiplication)
proof -
  let R = RealNumbers
  let A = RealAddition
  let M = RealMultiplication
  let P = PositiveReals
  let r = OrderOnReals
  let z = TheNeutralElement(R, A)
  have I:
    ring0(R, A, M)
```

```
    M {is commutative on} R
    P ⊆ R
    P {is closed under} A
    TheNeutralElement(R, A) ∉ P
    ∀a∈R. a ≠ z ⟶ (a ∈ P) Xor (GroupInv(R, A)(a) ∈ P)
    P {is closed under} M
    r = OrderFromPosSet(R, A, P)
    using real0.Real_ZF_1_L3 real_mult_commutative Real_ZF_1_2_L1
      real1.Real_ZF_1_2_L6 real1.Real_ZF_1_2_L9 OrderOnReals_def
    by auto
  then show IsAnOrdRing(R, A, M, r)
    by (rule ring0.ring_ord_by_positive_set)
  from I show r {is total on} R
    by (rule ring0.ring_ord_by_positive_set)
  from I show PositiveSet(R,A,r) = P
    by (rule ring0.ring_ord_by_positive_set)
  from I show HasNoZeroDivs(R,A,M)
    by (rule ring0.ring_ord_by_positive_set)
qed
```

All theorems proven in the `ring1` (about ordered rings), `group3` (about ordered groups) and `group1` (about groups) contexts are valid as applied to ordered real numbers with addition and (real) order.

```
lemma Real_ZF_1_2_L10: shows
  ring1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  IsAnOrdGroup(RealNumbers,RealAddition,OrderOnReals)
  group3(RealNumbers,RealAddition,OrderOnReals)
  OrderOnReals {is total on} RealNumbers
proof -
  show ring1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
    using reals_are_ord_ring OrdRing_ZF_1_L2 by simp
  then show
    IsAnOrdGroup(RealNumbers,RealAddition,OrderOnReals)
    group3(RealNumbers,RealAddition,OrderOnReals)
    OrderOnReals {is total on} RealNumbers
    using ring1.OrdRing_ZF_1_L4 by auto
qed
```

If $a = b$ or $b - a$ is positive, then a is less or equal b.

```
lemma (in real1) Real_ZF_1_2_L11: assumes A1: a∈ℝ  b∈ℝ and
  A3: a=b ∨ b-a ∈ PositiveReals
  shows a≤b
  using prems reals_are_ord_ring Real_ZF_1_2_L10
    group3.OrderedGroup_ZF_1_L30 by simp
```

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

```
lemma (in real1) Real_ZF_1_2_L12: assumes A1: f ∈ 𝒮  g ∈ 𝒮 and
  A2: f∼g ∨ (g + (-f)) ∈ 𝒮₊
```

```
  shows [f] ≤ [g]
proof -
  from A1 A2 have [f] = [g] ∨ [g]-[f] ∈ PositiveReals
    using Real_ZF_1_1_L5A Real_ZF_1_2_L2 Real_ZF_1_1_L4B
    by auto
  with A1 show [f] ≤ [g] using  Real_ZF_1_1_L3 Real_ZF_1_2_L11
    by simp
qed
```

Taking negative on both sides reverses the inequality, a case with an inverse on one side. Property of ordered groups.

```
lemma (in real1) Real_ZF_1_2_L13:
  assumes A1: a∈ℝ and A2: (-a) ≤ b
  shows (-b) ≤ a
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L5AG
  by simp
```

Real order is antisymmetric.

```
lemma (in real1) real_ord_antisym:
  assumes A1: a≤b  b≤a shows a=b
proof -
  from A1 have
    group3(RealNumbers,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,a⟩ ∈ OrderOnReals
    using Real_ZF_1_2_L10 by auto
  then show a=b by (rule group3.group_order_antisym)
qed
```

Real order is transitive.

```
lemma (in real1) real_ord_transitive: assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    group3(RealNumbers,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,c⟩ ∈ OrderOnReals
    using Real_ZF_1_2_L10 by auto
  then have ⟨a,c⟩ ∈ OrderOnReals
    by (rule group3.Group_order_transitive)
  then show a≤c by simp
qed
```

We can multiply both sides of an inequality by a nonnegative real number.

```
lemma (in real1) Real_ZF_1_2_L14:
  assumes a≤b and 0≤c
  shows
  a·c ≤ b·c
  c·a ≤ c·b
  using prems Real_ZF_1_2_L10 ring1.OrdRing_ZF_1_L9
```

```
  by auto
```

A special case of `Real_ZF_1_2_L14`: we can multiply an inequality by a real number.

```
lemma (in real1) Real_ZF_1_2_L14A:
  assumes A1: a≤b and A2: c∈ℝ+
  shows c·a ≤ c·b
  using prems Real_ZF_1_2_L10 ring1.OrdRing_ZF_1_L9A
  by simp
```

In the `real1` context notation $a \leq b$ implies that a and b are real numbers.

```
lemma (in real1) Real_ZF_1_2_L15: assumes a≤b shows a∈ℝ  b∈ℝ
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L4
  by auto
```

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

```
lemma (in real1) Real_ZF_1_2_L16: assumes a≤b
  shows 0 ≤ b-a
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L12A
  by simp
```

A sum of nonnegative elements is nonnegative.

```
lemma (in real1) Real_ZF_1_2_L17: assumes 0≤a 0≤b
  shows 0 ≤ a+b
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L12
  by simp
```

We can add sides of two inequalities

```
lemma (in real1) Real_ZF_1_2_L18: assumes a≤b  c≤d
  shows a+c ≤ b+d
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L5B
  by simp
```

The order on real is reflexive.

```
lemma (in real1) real_ord_refl: assumes a∈ℝ shows a≤a
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L3
  by simp
```

We can add a real number to both sides of an inequality.

```
lemma (in real1) add_num_to_ineq: assumes a≤b and c∈ℝ
  shows a+c ≤ b+c
  using prems Real_ZF_1_2_L10 IsAnOrdGroup_def by simp
```

We can put a number on the other side of an inequality, changing its sign.

```
lemma (in real1) Real_ZF_1_2_L19:
  assumes a∈ℝ  b∈ℝ and c ≤ a+b
  shows c-b ≤ a
```

```
  using prems  Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L9C
  by simp
```

What happens when one real number is not greater or equal than another?

```
lemma (in real1) Real_ZF_1_2_L20: assumes a∈ℝ  b∈ℝ and ¬(a≤b)
  shows b < a
proof -
  from prems have I:
    group3(ℝ,RealAddition,OrderOnReals)
    OrderOnReals {is total on} ℝ
    a∈ℝ  b∈ℝ  ¬(⟨a,b⟩ ∈ OrderOnReals)
    using Real_ZF_1_2_L10 by auto
  then have ⟨b,a⟩ ∈ OrderOnReals
    by (rule group3.OrderedGroup_ZF_1_L8)
  then have b ≤ a by simp
  moreover from I have a≠b by (rule group3.OrderedGroup_ZF_1_L8)
  ultimately show b < a by auto
qed
```

We can put a number on the other side of an inequality, changing its sign, version with a minus.

```
lemma (in real1) Real_ZF_1_2_L21:
  assumes a∈ℝ  b∈ℝ and c ≤ a-b
  shows c+b ≤ a
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L5J
  by simp
```

The order on reals is a relation on reals.

```
lemma (in real1) Real_ZF_1_2_L22: shows OrderOnReals ⊆ ℝ×ℝ
  using Real_ZF_1_2_L10 IsAnOrdGroup_def
  by simp
```

A set that is bounded above in the sense defined by order on reals is a subset of real numbers.

```
lemma (in real1) Real_ZF_1_2_L23:
  assumes A1: IsBoundedAbove(A,OrderOnReals)
  shows A ⊆ ℝ
  using A1 Real_ZF_1_2_L22 Order_ZF_3_L1A
  by blast
```

Properties of the maximum of three real numbers.

```
lemma (in real1) Real_ZF_1_2_L24:
  assumes A1: a∈ℝ  b∈ℝ  c∈ℝ
  shows
  Maximum(OrderOnReals,{a,b,c}) ∈ {a,b,c}
  Maximum(OrderOnReals,{a,b,c}) ∈ ℝ
  a ≤ Maximum(OrderOnReals,{a,b,c})
  b ≤ Maximum(OrderOnReals,{a,b,c})
```

```
  c ≤ Maximum(OrderOnReals,{a,b,c})
proof -
  have IsLinOrder(ℝ,OrderOnReals)
    using Real_ZF_1_2_L10 group3.group_ord_total_is_lin
    by simp
  with A1 show
    Maximum(OrderOnReals,{a,b,c}) ∈ {a,b,c}
    Maximum(OrderOnReals,{a,b,c}) ∈ ℝ
    a ≤ Maximum(OrderOnReals,{a,b,c})
    b ≤ Maximum(OrderOnReals,{a,b,c})
    c ≤ Maximum(OrderOnReals,{a,b,c})
    using Finite_ZF_1_L2A by auto
qed

lemma (in real1) real_strict_ord_transit:
  assumes A1: a≤b and A2: b<c
  shows a<c
proof -
  from A1 A2 have I:
    group3(ℝ,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  ⟨b,c⟩ ∈ OrderOnReals ∧ b≠c
    using Real_ZF_1_2_L10 by auto
  then have ⟨a,c⟩ ∈ OrderOnReals ∧ a≠c by (rule group3.group_strict_ord_transit)
  then show a<c by simp
qed
```

We can multiply a right hand side of an inequality between positive real numbers by a number that is greater than one.

```
lemma (in real1) Real_ZF_1_2_L25:
  assumes b ∈ ℝ+ and a≤b and 1<c
  shows a<b·c
  using prems reals_are_ord_ring Real_ZF_1_2_L10 ring1.OrdRing_ZF_3_L17
  by simp
```

We can move a real number to the other side of a strict inequality, changing its sign.

```
lemma (in real1) Real_ZF_1_2_L26:
  assumes a∈ℝ  b∈ℝ and  a-b < c
  shows a < c+b
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_1_L12B
  by simp
```

Real order is translation invariant.

```
lemma (in real1) real_ord_transl_inv:
  assumes a≤b and c∈ℝ
  shows c+a ≤ c+b
  using prems Real_ZF_1_2_L10 IsAnOrdGroup_def
  by simp
```

It is convenient to have the transitivity of the order on integers in the notation specific to real1 context. This may be confusing for the presentation readers: even though ≤ and ≤ are printed in the same way, they are different symbols in the source. In the real1 context the former denotes inequality between integers, and the latter denotes inequality between real numbers (classes of slopes). The next lemma is about transitivity of the order relation on integers.

```
lemma (in real1) int_order_transitive:
  assumes A1: a≤b  b≤c
  shows a≤c
proof -
  from A1 have
    ⟨a,b⟩ ∈ IntegerOrder and ⟨b,c⟩ ∈ IntegerOrder
    by auto
  then have ⟨a,c⟩ ∈ IntegerOrder
    by (rule Int_ZF_2_L5)
  then show a≤c by simp
qed
```

A property of nonempty subsets of real numbers that don't have a maximum: for any element we can find one that is (strictly) greater.

```
lemma (in real1) Real_ZF_1_2_L27:
  assumes A⊆ℝ and ¬HasAmaximum(OrderOnReals,A) and x∈A
  shows ∃y∈A. x<y
  using prems Real_ZF_1_2_L10 group3.OrderedGroup_ZF_2_L2B
  by simp
```

The next lemma shows what happens when one real number is not greater or equal than another.

```
lemma (in real1) Real_ZF_1_2_L28:
  assumes a∈ℝ  b∈ℝ and ¬(a≤b)
  shows b<a
proof -
  from prems have
    group3(ℝ,RealAddition,OrderOnReals)
    OrderOnReals {is total on} ℝ
    a∈ℝ  b∈ℝ  ⟨a,b⟩ ∉ OrderOnReals
    using Real_ZF_1_2_L10 by auto
  then have ⟨b,a⟩ ∈ OrderOnReals  ∧ b≠a
    by (rule group3.OrderedGroup_ZF_1_L8)
  then show b<a by simp
qed
```

If a real number is less than another, then the secon one can not be less or equal that the first.

```
lemma (in real1) Real_ZF_1_2_L29:
  assumes a<b shows ¬(b≤a)
```

```
proof -
  from prems have
    group3(ℝ,RealAddition,OrderOnReals)
    ⟨a,b⟩ ∈ OrderOnReals  a≠b
    using Real_ZF_1_2_L10 by auto
  then have ⟨b,a⟩ ∉ OrderOnReals
    by (rule group3.OrderedGroup_ZF_1_L8AA)
  then show ¬(b≤a) by simp
qed
```

28.4 Inverting reals

In this section we tackle the issue of existence of (multiplicative) inverses of real numbers and show that real numbers form an ordered field. We also restate here some facts specific to ordered fields that we need for the construction. The actual proofs of most of these facts can be found in Field_ZF.thy and OrderedField_ZF.thy

We rewrite the theorem from Int_ZF_2.thy that shows that for every positive slope we can find one that is almost equal and has an inverse.

```
lemma (in real1) pos_slopes_have_inv: assumes f ∈ 𝒮₊
  shows ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(int))
  using prems PositiveSlopes_def Slopes_def PositiveIntegers_def
    int1.pos_slope_has_inv SlopeOp1_def SlopeOp2_def
    BoundedIntMaps_def SlopeEquivalenceRel_def
  by simp
```

The set of real numbers we are constructing is an ordered field.

```
theorem (in real1) reals_are_ord_field: shows
  IsAnOrdField(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
proof -
  let R = RealNumbers
  let A = RealAddition
  let M = RealMultiplication
  let r = OrderOnReals
  have ring1(R,A,M,r) and 0 ≠ 1
    using reals_are_ord_ring OrdRing_ZF_1_L2 real_zero_not_one
    by auto
  moreover have M {is commutative on} R
    using real_mult_commutative by simp
  moreover have
    ∀a∈PositiveSet(R,A,r). ∃b∈R. a·b = 1
  proof
    fix a assume a ∈ PositiveSet(R,A,r)
    then obtain f where I: f∈𝒮₊ and II: a = [f]
      using reals_are_ord_ring Real_ZF_1_2_L2
      by auto
    then have ∃g∈𝒮. f∼g ∧ (∃h∈𝒮. g∘h ∼ id(int))
```

```
      using pos_slopes_have_inv by simp
    then obtain g where
      III: g∈𝒮 and IV: f∼g and V: ∃h∈𝒮. g∘h ∼ id(int)
      by auto
    from V obtain h where VII: h∈𝒮 and VIII: g∘h ∼ id(int)
      by auto
    from I III IV have [f] = [g]
      using Real_ZF_1_2_L1 Slopes_def Real_ZF_1_1_L5
      by auto
    with II III VII VIII have a·[h] = 1
      using Real_ZF_1_1_L4  Real_ZF_1_1_L5A real_one_cl_identity
      by simp
    with VII show ∃b∈R. a·b = 1 using Real_ZF_1_1_L3
      by auto
  qed
  ultimately show thesis using ring1.OrdField_ZF_1_L4
    by simp
qed
```

Reals form a field.

```
lemma reals_are_field:
  shows IsAfield(RealNumbers,RealAddition,RealMultiplication)
  using real1.reals_are_ord_field OrdField_ZF_1_L1A
  by simp
```

Theorem proven in `field0` and `field1` contexts are valid as applied to real numbers.

```
lemma field_cntxts_ok: shows
  field0(RealNumbers,RealAddition,RealMultiplication)
  field1(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  using reals_are_field real1.reals_are_ord_field
    Field_ZF_1_L2 OrdField_ZF_1_L2 by auto
```

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

```
lemma (in real1) Real_ZF_1_3_L1: assumes a ∈ ℝ₊
  shows a⁻¹ ∈ ℝ₊    a⁻¹ ∈ ℝ
  using prems field_cntxts_ok field1.OrdField_ZF_1_L8 PositiveSet_def
  by auto
```

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

```
lemma (in real1) Real_ZF_1_3_L2:
  assumes a ∈ ℝ₊ and a⁻¹ < b
  shows 1 < b·a
  using prems field_cntxts_ok field1.OrdField_ZF_2_L2
  by simp
```

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

```
lemma (in real1) Real_ZF_1_3_L3: assumes a<b
  shows (b-a)⁻¹ ∈ ℝ₊
  using prems field_cntxts_ok field1.OrdField_ZF_1_L9
  by simp
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse.

```
lemma (in real1) Real_ZF_1_3_L4:
  assumes A1: a∈ℝ  b∈ℝ₊ and A2: a·b < c
  shows a < c·b⁻¹
  using prems field_cntxts_ok field1.OrdField_ZF_2_L6
  by simp
```

We can put a positive factor on the other side of a strict inequality, changing it to its inverse, version with the product initially on the right hand side.

```
lemma (in real1) Real_ZF_1_3_L4A:
  assumes A1: b∈ℝ  c∈ℝ₊ and A2: a < b·c
  shows a·c⁻¹ < b
  using prems field_cntxts_ok field1.OrdField_ZF_2_L6A
  by simp
```

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with the product initially on the right hand side.

```
lemma (in real1) Real_ZF_1_3_L4B:
  assumes A1: b∈ℝ  c∈ℝ₊ and A2: a ≤ b·c
  shows a·c⁻¹ ≤ b
  using prems field_cntxts_ok field1.OrdField_ZF_2_L5A
  by simp
```

We can put a positive factor on the other side of an inequality, changing it to its inverse, version with the product initially on the left hand side.

```
lemma (in real1) Real_ZF_1_3_L4C:
  assumes A1: a∈ℝ  b∈ℝ₊ and A2: a·b ≤ c
  shows a ≤ c·b⁻¹
  using prems field_cntxts_ok field1.OrdField_ZF_2_L5
  by simp
```

A technical lemma about solving a strict inequality with three real numbers and inverse of a difference.

```
lemma (in real1) Real_ZF_1_3_L5:
  assumes a<b and (b-a)⁻¹ < c
  shows 1 + a·c < b·c
  using prems field_cntxts_ok field1.OrdField_ZF_2_L9
  by simp
```

We can multiply an inequality by the inverse of a positive number.

```
lemma (in real1) Real_ZF_1_3_L6:
```

```
  assumes a≤b  and c∈ℝ₊ shows a·c⁻¹ ≤ b·c⁻¹
  using prems field_cntxts_ok field1.OrdField_ZF_2_L3
  by simp
```

We can multiply a strict inequality by a positive number or its inverse.

```
lemma (in real1) Real_ZF_1_3_L7:
  assumes a<b  and c∈ℝ₊ shows
  a·c < b·c
  c·a < c·b
  a·c⁻¹ < b·c⁻¹
  using prems field_cntxts_ok field1.OrdField_ZF_2_L4
  by auto
```

An identity with three real numbers, inverse and cancelling.

```
lemma (in real1) Real_ZF_1_3_L8: assumesa∈ℝ  b∈ℝ b≠0  c∈ℝ
  shows a·b·(c·b⁻¹) = a·c
  using prems field_cntxts_ok field0.Field_ZF_2_L6
  by simp
```

28.5 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 $\mathtt{m}^R$ is a real number. Recall that in real1 context $\mathtt{m}^R$ denotes the class of the slope $n \mapsto m \cdot n$.

```
lemma (in real1) real_int_is_real: assumes m ∈ int
  shows m^R ∈ ℝ
  using prems int1.Int_ZF_2_5_L1 Real_ZF_1_1_L3 by simp
```

The negative of the real embedding of an integer is the embedding of the negative of the integer.

```
lemma (in real1) Real_ZF_1_4_L1: assumes m ∈ int
  shows (-m)^R = -(m^R)
  using prems int1.Int_ZF_2_5_L3 int1.Int_ZF_2_5_L1 Real_ZF_1_1_L4A
  by simp
```

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

```
lemma (in real1) Real_ZF_1_4_L1A: assumes m ∈ int  k ∈ int
  shows m^R + k^R = ((m+k)^R)
  using prems int1.Int_ZF_2_5_L1 SlopeOp1_def int1.Int_ZF_2_5_L3A
    Real_ZF_1_1_L4 by simp
```

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

```
lemma (in real1) Real_ZF_1_4_L1B: assumes A1: m ∈ int  k ∈ int
  shows m^R - k^R = (m-k)^R
```

```
proof -
  from A1 have (-k) ∈ int using int0.Int_ZF_1_1_L4
    by simp
  with A1 have (m-k)^R = m^R + (-k)^R
    using Real_ZF_1_4_L1A by simp
  with A1 show m^R - k^R = (m-k)^R
    using Real_ZF_1_4_L1 by simp
qed
```

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

```
lemma (in real1) Real_ZF_1_4_L1C: assumes m ∈ int  k ∈ int
  shows m^R · k^R = (m·k)^R
  using prems int1.Int_ZF_2_5_L1 SlopeOp2_def int1.Int_ZF_2_5_L3B
    Real_ZF_1_1_L4 by simp
```

For any real numbers there is an integer whose real version is greater or equal.

```
lemma (in real1) Real_ZF_1_4_L2: assumes A1: a∈ℝ
  shows ∃m∈int. a ≤ m^R
proof -
  from A1 obtain f where I: f∈𝒮 and II: a = [f]
    using Real_ZF_1_1_L3A by auto
  then have ∃m∈int. ∃g∈𝒮.
    {⟨n,m·n⟩ . n ∈ int} ∼ g ∧ (f∼g ∨ (g + (-f)) ∈ 𝒮₊)
    using int1.Int_ZF_2_5_L2 Slopes_def SlopeOp1_def
      BoundedIntMaps_def SlopeEquivalenceRel_def
      PositiveIntegers_def PositiveSlopes_def
    by simp
  then obtain m g where III: m∈int and IV: g∈𝒮 and
   {⟨n,m·n⟩ . n ∈ int} ∼ g ∧ (f∼g ∨ (g + (-f)) ∈ 𝒮₊)
    by auto
  then have m^R = [g] and f ∼ g ∨ (g + (-f)) ∈ 𝒮₊
    using Real_ZF_1_1_L5A by auto
  with I II IV have a ≤ m^R using Real_ZF_1_2_L12
    by simp
  with III show ∃m∈int. a ≤ m^R by auto
qed
```

For any real numbers there is an integer whose real version (embedding) is less or equal.

```
lemma (in real1) Real_ZF_1_4_L3: assumes A1: a∈ℝ
  shows {m ∈ int. m^R ≤ a} ≠ 0
proof -
  from A1 have (-a) ∈ ℝ using Real_ZF_1_1_L8
    by simp
  then obtain m where I: m∈int and II: (-a) ≤ m^R
    using Real_ZF_1_4_L2 by auto
  let k = GroupInv(int,IntegerAddition)(m)
```

```
  from A1 I II have k ∈ int and k^R ≤ a
    using Real_ZF_1_2_L13 Real_ZF_1_4_L1 int0.Int_ZF_1_1_L4
    by auto
  then show thesis by auto
qed
```

Embeddings of two integers are equal only if the integers are equal.

```
lemma (in real1) Real_ZF_1_4_L4:
  assumes A1: m ∈ int  k ∈ int and A2: m^R = k^R
  shows m=k
proof -
  let r = {⟨n, IntegerMultiplication ⟨m, n⟩⟩ . n ∈ int}
  let s = {⟨n, IntegerMultiplication ⟨k, n⟩⟩ . n ∈ int}
  from A1 A2 have r ∼ s
    using int1.Int_ZF_2_5_L1 AlmostHoms_def Real_ZF_1_1_L5
    by simp
  with A1 have
    m ∈ int  k ∈ int
    ⟨r,s⟩ ∈ QuotientGroupRel(AlmostHoms(int, IntegerAddition),
    AlHomOp1(int, IntegerAddition),FinRangeFunctions(int, int))
    using SlopeEquivalenceRel_def Slopes_def SlopeOp1_def
      BoundedIntMaps_def by auto
  then show m=k by (rule int1.Int_ZF_2_5_L6)
qed
```

The embedding of integers preserves the order.

```
lemma (in real1) Real_ZF_1_4_L5: assumes A1: m≤k
  shows m^R ≤ k^R
proof -
  let r = {⟨n, m·n⟩ . n ∈ int}
  let s = {⟨n, k·n⟩ . n ∈ int}
  from A1 have r ∈ 𝒮  s ∈ 𝒮
    using int0.Int_ZF_2_L1A int1.Int_ZF_2_5_L1 by auto
  moreover from A1 have r ∼ s ∨ s + (-r)  ∈ 𝒮+
    using Slopes_def SlopeOp1_def BoundedIntMaps_def SlopeEquivalenceRel_def
      PositiveIntegers_def PositiveSlopes_def
      int1.Int_ZF_2_5_L4 by simp
  ultimately show m^R ≤ k^R using Real_ZF_1_2_L12
    by simp
qed
```

The embedding of integers preserves the strict order.

```
lemma (in real1) Real_ZF_1_4_L5A: assumes A1: m≤k  m≠k
  shows m^R < k^R
proof -
  from A1 have m^R ≤ k^R using Real_ZF_1_4_L5
    by simp
  moreover
  from A1 have T: m ∈ int  k ∈ int
```

```
    using int0.Int_ZF_2_L1A by auto
  with A1 have m^R ≠ k^R using Real_ZF_1_4_L4
    by auto
  ultimately show m^R < k^R by simp
qed
```

For any real number there is a positive integer whose real version is (strictly) greater. This is Lemma 14 i) in [2].

```
lemma (in real1) Arthan_Lemma14i: assumes A1: a∈ℝ
  shows ∃n∈ℤ+. a < n^R
proof -
  from A1 obtain m where I: m∈int and II: a ≤ m^R
    using Real_ZF_1_4_L2 by auto
  let n = GreaterOf(IntegerOrder,1_Z,m) + 1_Z
  from I have T: n ∈ℤ+ and m ≤ n  m≠n
    using int0.Int_ZF_1_5_L7B by auto
  then have III: m^R < n^R
    using Real_ZF_1_4_L5A by simp
  with II have a < n^R by (rule real_strict_ord_transit)
  with T show thesis by auto
qed
```

If one embedding is less or equal than another, then the integers are also less or equal.

```
lemma (in real1) Real_ZF_1_4_L6:
  assumes A1: k ∈ int  m ∈ int and A2: m^R ≤ k^R
  shows m≤k
proof -
  { assume A3: ⟨m,k⟩ ∉ IntegerOrder
    with A1 have ⟨k,m⟩ ∈ IntegerOrder
      by (rule int0.Int_ZF_2_L19)
    then have k^R ≤ m^R using Real_ZF_1_4_L5
      by simp
    with A2 have m^R = k^R by (rule real_ord_antisym)
    with A1 have k = m using Real_ZF_1_4_L4
      by auto
    moreover from A1 A3 have k≠m by (rule int0.Int_ZF_2_L19)
    ultimately have False by simp
  } then show m≤k by auto
qed
```

The floor function is well defined and has expected properties.

```
lemma (in real1) Real_ZF_1_4_L7: assumes A1: a∈ℝ
  shows
  IsBoundedAbove({m ∈ int. m^R ≤ a},IntegerOrder)
  {m ∈ int. m^R ≤ a} ≠ 0
  ⌊a⌋ ∈ int
  ⌊a⌋^R ≤ a
```

```
proof -
  let A = {m ∈ int. m^R ≤ a}
  from A1 obtain K where I: K∈int and II: a ≤ (K^R)
    using Real_ZF_1_4_L2 by auto
  { fix n assume n ∈ A
    then have III: n ∈ int and IV: n^R ≤ a
      by auto
    from IV II have (n^R) ≤ (K^R)
      by (rule real_ord_transitive)
    with I III have n≤K using Real_ZF_1_4_L6
      by simp
  } then have ∀n∈A. ⟨n,K⟩ ∈ IntegerOrder
    by simp
  then show IsBoundedAbove(A,IntegerOrder)
    by (rule Order_ZF_3_L10)
  moreover from A1 show A ≠ 0 using Real_ZF_1_4_L3
    by simp
  ultimately have Maximum(IntegerOrder,A) ∈ A
    by (rule int0.int_bounded_above_has_max)
  then show ⌊a⌋ ∈ int   ⌊a⌋^R ≤ a by auto
qed
```

Every integer whose embedding is less or equal a real number a is less or equal than the floor of a.

```
lemma (in real1) Real_ZF_1_4_L8:
  assumes A1: m ∈ int and A2: m^R ≤ a
  shows m ≤ ⌊a⌋
proof -
  let A = {m ∈ int. m^R ≤ a}
  from A2 have IsBoundedAbove(A,IntegerOrder) and A≠0
    using Real_ZF_1_2_L15 Real_ZF_1_4_L7 by auto
  then have ∀x∈A. ⟨x,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule int0.int_bounded_above_has_max)
  with A1 A2 show m ≤ ⌊a⌋ by simp
qed
```

Integer zero and one embed as real zero and one.

```
lemma (in real1) int_0_1_are_real_zero_one:
  shows 0_Z^R = 0   1_Z^R = 1
  using int1.Int_ZF_2_5_L7 BoundedIntMaps_def
    real_one_cl_identity real_zero_cl_bounded_map
  by auto
```

Integer two embeds as the real two.

```
lemma (in real1) int_two_is_real_two: shows 2_Z^R = 2
proof -
  have 2_Z^R = 1_Z^R + 1_Z^R
    using int0.int_zero_one_are_int Real_ZF_1_4_L1A
    by simp
```

```
  also have ... = 2 using int_0_1_are_real_zero_one
    by simp
  finally show 2_Z^R = 2 by simp
qed
```

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

```
lemma (in real1) int_pos_is_real_pos: assumes A1: p∈ℤ+
  shows
  p^R ∈ ℝ
  0 ≤ p^R
  p^R ∈ ℝ+
proof -
  from A1 have I: p ∈ int  0_Z ≤ p  0_Z ≠ p
    using PositiveSet_def by auto
  then have p^R ∈ ℝ  0_Z^R ≤ p^R
    using real_int_is_real Real_ZF_1_4_L5 by auto
  then show p^R ∈ ℝ  0 ≤ p^R
    using int_0_1_are_real_zero_one by auto
  moreover have 0 ≠ p^R
  proof -
    { assume 0 = p^R
      with I have False using int_0_1_are_real_zero_one
        int0.int_zero_one_are_int Real_ZF_1_4_L4 by auto
    } then show 0 ≠ p^R by auto
  qed
  ultimately show p^R ∈ ℝ+ using PositiveSet_def
    by simp
qed
```

The ordered field of reals we are constructing is archimedean, i.e., if x, y are its elements with y positive, then there is a positive integer M such that $x < M^R y$. This is Lemma 14 ii) in [2].

```
lemma (in real1) Arthan_Lemma14ii: assumes A1: x∈ℝ  y ∈ ℝ+
  shows ∃M∈ℤ+. x < M^R·y
proof -
  from A1 have
    ∃C∈ℤ+. x < C^R and ∃D∈ℤ+. y^-1 < D^R
    using Real_ZF_1_3_L1 Arthan_Lemma14i by auto
  then obtain C D where
    I: C∈ℤ+ and II: x < C^R and
    III: D∈ℤ+ and IV: y^-1 < D^R
    by auto
  let M = C·D
  from I III have
    T: M ∈ ℤ+  C^R ∈ ℝ  D^R ∈ ℝ
    using int0.pos_int_closed_mul_unfold PositiveSet_def real_int_is_real
    by auto
  with A1 I III have C^R·(D^R·y) = M^R·y
    using PositiveSet_def Real_ZF_1_L6A Real_ZF_1_4_L1C
```

```
    by simp
  moreover from A1 I II IV have
    x < C^R·(D^R·y)
    using int_pos_is_real_pos Real_ZF_1_3_L2 Real_ZF_1_2_L25
    by auto
  ultimately have x < M^R·y
    by auto
  with T show thesis by auto
qed
```

Taking the floor function preserves the order.

```
lemma (in real1) Real_ZF_1_4_L9: assumes A1: a≤b
  shows ⌊a⌋ ≤ ⌊b⌋
proof -
  from A1 have T: a∈ℝ using Real_ZF_1_2_L15
    by simp
  with A1 have ⌊a⌋^R ≤ a and a≤b
    using Real_ZF_1_4_L7 by auto
  then have ⌊a⌋^R ≤ b by (rule real_ord_transitive)
  moreover from T have ⌊a⌋ ∈ int using Real_ZF_1_4_L7
    by simp
 ultimately show ⌊a⌋ ≤ ⌊b⌋ using Real_ZF_1_4_L8
   by simp
qed
```

If S is bounded above and p is a positive 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∈ℤ+
  shows
  IsBoundedAbove({⌊p^R·x⌋. x∈S},IntegerOrder)
  Γ(S,p) ∈ {⌊p^R·x⌋. x∈S}
  Γ(S,p) ∈ int
proof -
  let A = {⌊p^R·x⌋. x∈S}
  from A1 obtain X where I: ∀x∈S. x≤X
    using IsBoundedAbove_def by auto
  { fix m assume m ∈ A
    then obtain x where x∈S and II: m = ⌊p^R·x⌋
      by auto
    with I have x≤X by simp
    moreover from A2 have 0 ≤ p^R using int_pos_is_real_pos
      by simp
    ultimately have p^R·x ≤ p^R·X using Real_ZF_1_2_L14
      by simp
    with II have m ≤ ⌊p^R·X⌋ using Real_ZF_1_4_L9
      by simp
  } then have ∀m∈A. ⟨m,⌊p^R·X⌋⟩ ∈ IntegerOrder
    by auto
```

```
  then show II: IsBoundedAbove(A,IntegerOrder)
    by (rule Order_ZF_3_L10)
  moreover from A1 have III: A ≠ 0 by simp
  ultimately have Maximum(IntegerOrder,A) ∈ A
    by (rule int0.int_bounded_above_has_max)
  moreover from II III have Maximum(IntegerOrder,A) ∈ int
    by (rule int0.int_bounded_above_has_max)
  ultimately show Γ(S,p) ∈ {⌊p^R·x⌋. x∈S} and Γ(S,p) ∈ int
    by auto
qed
```

If p is a positive integer, then for all $s \in S$ the floor of $p \cdot x$ is not greater that $\Gamma(S, p)$.

```
lemma (in real1) Real_ZF_1_4_L11:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and A2: x∈S and A3: p∈ℤ+
  shows ⌊p^R·x⌋ ≤ Γ(S,p)
proof -
  let A = {⌊p^R·x⌋. x∈S}
  from A2 have S≠0 by auto
  with A1 A3 have IsBoundedAbove(A,IntegerOrder)  A ≠ 0
    using  Real_ZF_1_4_L10 by auto
  then have ∀x∈A. ⟨x,Maximum(IntegerOrder,A)⟩ ∈ IntegerOrder
    by (rule int0.int_bounded_above_has_max)
  with A2 show ⌊p^R·x⌋ ≤ Γ(S,p) by simp
qed
```

The candidate for supremum is an integer mapping with values given by Γ.

```
lemma (in real1) Real_ZF_1_4_L12:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: g = {⟨p,Γ(S,p)⟩. p∈ℤ+}
  shows
  g : ℤ+→int
  ∀n∈ℤ+. g(n) = Γ(S,n)
proof -
  from A1 have ∀n∈ℤ+. Γ(S,n) ∈ int using Real_ZF_1_4_L10
    by simp
  with A2 show I: g : ℤ+→int using ZF_fun_from_total by simp
  { fix n assume n∈ℤ+
    with A2 I have g(n) = Γ(S,n) using ZF_fun_from_tot_val
      by simp
  } then show ∀n∈ℤ+. g(n) = Γ(S,n) by simp
qed
```

Every integer is equal to the floor of its embedding.

```
lemma (in real1) Real_ZF_1_4_L14: assumes A1: m ∈ int
  shows ⌊m^R⌋ = m
proof -
  let A = {n ∈ int. n^R ≤ m^R }
  have antisym(IntegerOrder) using int0.Int_ZF_2_L4
```

```
    by simp
  moreover from A1 have m ∈ A
    using real_int_is_real real_ord_refl by auto
  moreover from A1 have ∀n ∈ A. ⟨n,m⟩ ∈ IntegerOrder
    using Real_ZF_1_4_L6 by auto
  ultimately show ⌊m^R⌋ = m using Order_ZF_4_L14
    by auto
qed
```

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

```
lemma (in real1) floor_01_is_zero_one: shows
  ⌊0⌋ = 0_Z    ⌊1⌋ = 1_Z
proof -
  have ⌊(0_Z)^R⌋ = 0_Z and ⌊(1_Z)^R⌋ = 1_Z
    using int0.int_zero_one_are_int Real_ZF_1_4_L14
    by auto
  then show ⌊0⌋ = 0_Z and  ⌊1⌋ = 1_Z
    using int_0_1_are_real_zero_one
    by auto
qed
```

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

```
lemma (in real1) floor_2_is_two: shows ⌊2⌋ = 2_Z
proof -
  have ⌊(2_Z)^R⌋ = 2_Z
    using int0.int_two_three_are_int Real_ZF_1_4_L14
    by simp
  then show ⌊2⌋ = 2_Z using int_two_is_real_two
    by simp
qed
```

Floor of a product of embeddings of integers is equal to the product of integers.

```
lemma (in real1) Real_ZF_1_4_L14A: assumes A1: m ∈ int  k ∈ int
  shows  ⌊m^R·k^R⌋ = m·k
proof -
  from A1 have T: m·k ∈ int
    using int0.Int_ZF_1_1_L5 by simp
  from A1 have ⌊m^R·k^R⌋ = ⌊(m·k)^R⌋ using Real_ZF_1_4_L1C
    by simp
  with T show ⌊m^R·k^R⌋ = m·k using Real_ZF_1_4_L14
    by simp
qed
```

Floor of the sum of a number and the embedding of an integer is the floor of the number plus the integer.

```
lemma (in real1) Real_ZF_1_4_L15: assumes A1: x∈ℝ and A2: p ∈ int
  shows ⌊x + p^R⌋ = ⌊x⌋ + p
```

```
proof -
  let A = {n ∈ int. n^R ≤ x + p^R}
  have antisym(IntegerOrder) using int0.Int_ZF_2_L4
    by simp
  moreover have ⌊x⌋ + p ∈ A
  proof -
    from A1 A2 have ⌊x⌋^R ≤ x and p^R ∈ ℝ
      using Real_ZF_1_4_L7 real_int_is_real by auto
    then have ⌊x⌋^R + p^R ≤ x + p^R
      using add_num_to_ineq by simp
    moreover from A1 A2 have (⌊x⌋ + p)^R = ⌊x⌋^R + p^R
      using Real_ZF_1_4_L7 Real_ZF_1_4_L1A by simp
    ultimately have (⌊x⌋ + p)^R ≤ x + p^R
      by simp
    moreover from A1 A2 have ⌊x⌋ + p ∈ int
      using Real_ZF_1_4_L7 int0.Int_ZF_1_1_L5 by simp
    ultimately show ⌊x⌋ + p ∈ A by auto
  qed
  moreover have ∀n∈A. n ≤ ⌊x⌋ + p
  proof
    fix n assume n∈A
    then have I: n ∈ int and n^R ≤ x + p^R
      by auto
    with A1 A2 have n^R - p^R ≤ x
      using real_int_is_real Real_ZF_1_2_L19
      by simp
    with A2 I have ⌊(n-p)^R⌋ ≤ ⌊x⌋
      using Real_ZF_1_4_L1B Real_ZF_1_4_L9
      by simp
    moreover
    from A2 I have n-p ∈ int
      using int0.Int_ZF_1_1_L5 by simp
    then have ⌊(n-p)^R⌋ = n-p
      using Real_ZF_1_4_L14 by simp
    ultimately have n-p ≤ ⌊x⌋
      by simp
    with A2 I show n ≤ ⌊x⌋ + p
      using int0.Int_ZF_2_L9C by simp
  qed
  ultimately show ⌊x + p^R⌋ = ⌊x⌋ + p
    using Order_ZF_4_L14 by auto
qed
```

Floor of the difference of a number and the embedding of an integer is the floor of the number minus the integer.

```
lemma (in real1) Real_ZF_1_4_L16: assumes A1: x∈ℝ and A2: p ∈ int
  shows ⌊x - p^R⌋ = ⌊x⌋ - p
proof -
  from A2 have ⌊x - p^R⌋ = ⌊x + (-p)^R⌋
```

```
    using Real_ZF_1_4_L1 by simp
  with A1 A2 show ⌊x - p^R⌋ = ⌊x⌋ - p
    using int0.Int_ZF_1_1_L4 Real_ZF_1_4_L15 by simp
qed
```

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

```
lemma (in real1) Real_ZF_1_4_L17: assumes m ∈ int  n ∈ int
  shows ⌊(m^R) + n^R⌋ = m + n
  using prems real_int_is_real Real_ZF_1_4_L15 Real_ZF_1_4_L14
  by simp
```

A lemma about adding one to floor.

```
lemma (in real1) Real_ZF_1_4_L17A: assumes A1: a∈ℝ
  shows 1 + ⌊a⌋^R = (1_Z + ⌊a⌋)^R
proof -
  have 1 + ⌊a⌋^R = 1_Z^R + ⌊a⌋^R
    using int_0_1_are_real_zero_one by simp
  with A1 show 1 + ⌊a⌋^R = (1_Z + ⌊a⌋)^R
    using int0.int_zero_one_are_int Real_ZF_1_4_L7 Real_ZF_1_4_L1A
    by simp
qed
```

The difference between the a number and the embedding of its floor is (strictly) less than one.

```
lemma (in real1) Real_ZF_1_4_L17B: assumes A1: a∈ℝ
  shows
  a - ⌊a⌋^R < 1
  a < (1_Z + ⌊a⌋)^R
proof -
  from A1 have T1: ⌊a⌋ ∈ int  ⌊a⌋^R ∈ ℝ and
    T2: 1 ∈ ℝ  a -  ⌊a⌋^R ∈ ℝ
    using Real_ZF_1_4_L7 real_int_is_real Real_ZF_1_L6 Real_ZF_1_L4
    by auto
  { assume 1 ≤ a -  ⌊a⌋^R
    with A1 T1 have ⌊1_Z^R + ⌊a⌋^R⌋ ≤ ⌊a⌋
      using Real_ZF_1_2_L21 Real_ZF_1_4_L9 int_0_1_are_real_zero_one
      by simp
    with T1 have False
      using int0.int_zero_one_are_int Real_ZF_1_4_L17
      int0.Int_ZF_1_2_L3AA by simp
  } then have I: ¬(1 ≤ a - ⌊a⌋^R) by auto
  with T2 show II: a - ⌊a⌋^R < 1
    by (rule Real_ZF_1_2_L20)
   with A1 T1 I II have
    a < 1 + ⌊a⌋^R
    using Real_ZF_1_2_L26 by simp
  with A1 show a < (1_Z + ⌊a⌋)^R
    using Real_ZF_1_4_L17A by simp
qed
```

The next lemma corresponds to Lemma 14 iii) in [2]. It says that we can find a rational number between any two different real numbers.

```
lemma (in real1) Arthan_Lemma14iii: assumes A1: x<y
  shows ∃M∈int. ∃N∈ℤ+.  x·N^R < M^R ∧ M^R < y·N^R
proof -
  from A1 have (y-x)^-1 ∈ ℝ+ using Real_ZF_1_3_L3
    by simp
  then have
    ∃N∈ℤ+. (y-x)^-1 < N^R
    using Arthan_Lemma14i PositiveSet_def by simp
  then obtain N where I: N∈ℤ+ and II: (y-x)^-1 < N^R
    by auto
  let M = 1_Z + ⌊x·N^R⌋
  from A1 I have
    T1: x∈ℝ  N^R ∈ ℝ  N^R ∈ ℝ+  x·N^R ∈ ℝ
    using Real_ZF_1_2_L15 PositiveSet_def real_int_is_real
      Real_ZF_1_L6 int_pos_is_real_pos by auto
  then have T2: M ∈ int using
    int0.int_zero_one_are_int Real_ZF_1_4_L7 int0.Int_ZF_1_1_L5
    by simp
  from T1 have III: x·N^R < M^R
    using Real_ZF_1_4_L17B by simp
  from T1 have (1 + ⌊x·N^R⌋^R) ≤ (1 + x·N^R)
    using Real_ZF_1_4_L7  Real_ZF_1_L4 real_ord_transl_inv
    by simp
  with T1 have M^R ≤ (1 + x·N^R)
    using Real_ZF_1_4_L17A by simp
  moreover from A1 II have (1 + x·N^R) < y·N^R
    using Real_ZF_1_3_L5 by simp
  ultimately have M^R < y·N^R
    by (rule real_strict_ord_transit)
  with I T2 III show thesis by auto
qed
```

Some estimates for the homomorphism difference of the floor function.

```
lemma (in real1) Real_ZF_1_4_L18: assumes A1: x∈ℝ  y∈ℝ
  shows
  abs(⌊x+y⌋ - ⌊x⌋ - ⌊y⌋) ≤ 2_Z
proof -
  from A1 have T:
    ⌊x⌋^R ∈ ℝ  ⌊y⌋^R ∈ ℝ
    x+y - (⌊x⌋^R) ∈ ℝ
     using Real_ZF_1_4_L7 real_int_is_real Real_ZF_1_L6
     by auto
  from A1 have
    0 ≤ x - (⌊x⌋^R) + (y - (⌊y⌋^R))
    x -  (⌊x⌋^R) + (y - (⌊y⌋^R)) ≤ 2
    using Real_ZF_1_4_L7 Real_ZF_1_2_L16 Real_ZF_1_2_L17
      Real_ZF_1_4_L17B Real_ZF_1_2_L18 by auto
```

```
  moreover from A1 T have
    x - (⌊x⌋^R) + (y - (⌊y⌋^R)) = x+y - (⌊x⌋^R) - (⌊y⌋^R)
    using Real_ZF_1_L7A by simp
  ultimately have
    0 ≤ x+y - (⌊x⌋^R) - (⌊y⌋^R)
    x+y - (⌊x⌋^R) - (⌊y⌋^R) ≤ 2
    by auto
  then have
    ⌊0⌋ ≤ ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋
    ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋ ≤ ⌊2⌋
    using Real_ZF_1_4_L9 by auto
  then have
    0_Z  ≤ ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋
    ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋ ≤ 2_Z
    using floor_01_is_zero_one floor_2_is_two by auto
  moreover from A1 have
    ⌊x+y - (⌊x⌋^R) - (⌊y⌋^R)⌋ = ⌊x+y⌋ - ⌊x⌋ - ⌊y⌋
    using Real_ZF_1_L6 Real_ZF_1_4_L7 real_int_is_real Real_ZF_1_4_L16
    by simp
  ultimately have
    0_Z ≤ ⌊x+y⌋ - ⌊x⌋ - ⌊y⌋
    ⌊x+y⌋ - ⌊x⌋ - ⌊y⌋ ≤ 2_Z
    by auto
  then show abs(⌊x+y⌋ - ⌊x⌋ - ⌊y⌋) ≤ 2_Z
    using int0.Int_ZF_2_L16 by simp
qed
```

Suppose $S \neq \emptyset$ is bounded above and $\Gamma(S,m) = \lfloor m^R \cdot x \rfloor$ for some positive integer m and $x \in S$. Then if $y \in S, x \leq y$ we also have $\Gamma(S,m) = \lfloor m^R \cdot y \rfloor$.

```
lemma (in real1) Real_ZF_1_4_L20:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: n∈ℤ₊ x∈S and
  A3: Γ(S,n) = ⌊n^R·x⌋ and
  A4: y∈S  x≤y
  shows Γ(S,n) = ⌊n^R·y⌋
proof -
  from A2 A4 have ⌊n^R·x⌋ ≤ ⌊(n^R)·y⌋
    using int_pos_is_real_pos Real_ZF_1_2_L14 Real_ZF_1_4_L9
    by simp
  with A3 have ⟨Γ(S,n),⌊(n^R)·y⌋⟩ ∈ IntegerOrder
    by simp
  moreover from A1 A2 A4 have ⟨⌊n^R·y⌋,Γ(S,n)⟩ ∈ IntegerOrder
    using Real_ZF_1_4_L11 by simp
  ultimately show Γ(S,n) = ⌊n^R·y⌋
    by (rule int0.Int_ZF_2_L3)
qed
```

The homomorphism difference of $n \mapsto \Gamma(S,n)$ is bounded by 2 on positive integers.

```
lemma (in real1) Real_ZF_1_4_L21:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: m∈ℤ₊  n∈ℤ₊
  shows abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) ≤  2_Z
proof -
  from A2 have T: m+n ∈ ℤ₊ using int0.pos_int_closed_add_unfolded
    by simp
  with A1 A2 have
    Γ(S,m) ∈ {⌊m^R·x⌋. x∈S} and
    Γ(S,n) ∈ {⌊n^R·x⌋. x∈S} and
    Γ(S,m+n) ∈ {⌊(m+n)^R·x⌋. x∈S}
    using Real_ZF_1_4_L10 by auto
  then obtain a b c where I: a∈S  b∈S  c∈S
    and II:
    Γ(S,m) = ⌊m^R·a⌋
    Γ(S,n) = ⌊n^R·b⌋
    Γ(S,m+n) = ⌊(m+n)^R·c⌋
    by auto
  let d = Maximum(OrderOnReals,{a,b,c})
  from A1 I have a∈ℝ  b∈ℝ  c∈ℝ
    using Real_ZF_1_2_L23 by auto
  then have IV:
    d ∈ {a,b,c}
    d ∈ ℝ
    a ≤ d
    b ≤ d
    c ≤ d
    using Real_ZF_1_2_L24 by auto
  with I have V: d ∈ S by auto
  from A1 T I II IV V have Γ(S,m+n) = ⌊(m+n)^R·d⌋
    using Real_ZF_1_4_L20 by blast
  also from A2 have ... = ⌊((m^R)+(n^R))·d⌋
    using Real_ZF_1_4_L1A PositiveSet_def by simp
  also from A2 IV have ... = ⌊(m^R)·d + (n^R)·d⌋
    using PositiveSet_def real_int_is_real Real_ZF_1_L7
    by simp
  finally have  Γ(S,m+n) =  ⌊(m^R)·d + (n^R)·d⌋
    by simp
  moreover from A1 A2 I II IV V have Γ(S,m) = ⌊m^R·d⌋
    using Real_ZF_1_4_L20 by blast
  moreover from A1 A2 I II IV V have  Γ(S,n) = ⌊n^R·d⌋
    using Real_ZF_1_4_L20 by blast
  moreover from A1 T I II IV V have Γ(S,m+n) = ⌊(m+n)^R·d⌋
    using Real_ZF_1_4_L20 by blast
  ultimately have abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) =
    abs(⌊(m^R)·d + (n^R)·d⌋ - ⌊m^R·d⌋ - ⌊n^R·d⌋)
    by simp
  with A2 IV show
    abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) ≤  2_Z
```

```
    using PositiveSet_def real_int_is_real Real_ZF_1_L6
      Real_ZF_1_4_L18 by simp
qed
```

The next lemma provides sufficient condition for an odd function to be an almost homomorphism. It says for odd functions we only need to check that the homomorphism difference (denoted δ in the `real1` context is bounded on positive integers. This is really proven in `Int_ZF_2.thy`, but we restate it here for convenience. Recall from `Group_ZF_3.thy` that `OddExtension` of a function defined on the set of positive elements (of an ordered group) is the only odd function that is equal to the given one when restricted to positive elements.

```
lemma (in real1) Real_ZF_1_4_L21A:
  assumes A1: f:ℤ₊→int  ∀a∈ℤ₊. ∀b∈ℤ₊. abs(δ(f,a,b)) ≤ L
  shows OddExtension(int,IntegerAddition,IntegerOrder,f) ∈ 𝒮
  using A1 int1.Int_ZF_2_1_L24 by auto
```

The candidate for (a representant of) the supremum of a nonempty bounded above set is a slope.

```
lemma (in real1) Real_ZF_1_4_L22:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and
  A2: g = {⟨p,Γ(S,p)⟩. p∈ℤ₊}
  shows OddExtension(int,IntegerAddition,IntegerOrder,g) ∈ 𝒮
proof -
  from A1 A2 have g: ℤ₊→int by (rule Real_ZF_1_4_L12)
  moreover have ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ 2_Z
  proof -
    { fix m n assume A3: m∈ℤ₊  n∈ℤ₊
      then have m+n ∈ ℤ₊  m∈ℤ₊  n∈ℤ₊
        using int0.pos_int_closed_add_unfolded
        by auto
      moreover from A1 A2 have ∀n∈ℤ₊. g(n) = Γ(S,n)
        by (rule Real_ZF_1_4_L12)
      ultimately have δ(g,m,n) = Γ(S,m+n) - Γ(S,m) - Γ(S,n)
        by simp
      moreover from A1 A3 have
        abs(Γ(S,m+n) - Γ(S,m) - Γ(S,n)) ≤  2_Z
        by (rule Real_ZF_1_4_L21)
      ultimately have abs(δ(g,m,n)) ≤ 2_Z
        by simp
    } then show ∀m∈ℤ₊. ∀n∈ℤ₊. abs(δ(g,m,n)) ≤ 2_Z
      by simp
  qed
  ultimately show thesis by (rule Real_ZF_1_4_L21A)
qed
```

A technical lemma used in the proof that all elements of S are less or equal than the candidate for supremum of S.

```
lemma (in real1) Real_ZF_1_4_L23:
  assumes A1: f ∈ 𝒮 and A2: N ∈ int  M ∈ int and
  A3: ∀n∈ℤ₊. M·n ≤ f(N·n)
  shows M^R ≤ [f]·(N^R)
proof -
  let M^S = {⟨n, M·n⟩ . n ∈ int}
  let N^S = {⟨n, N·n⟩ . n ∈ int}
  from A1 A2 have T: M^S ∈ 𝒮  N^S ∈ 𝒮  f∘N^S ∈ 𝒮
    using int1.Int_ZF_2_5_L1 int1.Int_ZF_2_1_L11 SlopeOp2_def
    by auto
  moreover from A1 A2 A3 have M^S ∼ f∘N^S ∨ f∘N^S + (-M^S) ∈ 𝒮₊
    using int1.Int_ZF_2_5_L8 SlopeOp2_def SlopeOp1_def Slopes_def
      BoundedIntMaps_def SlopeEquivalenceRel_def PositiveIntegers_def
      PositiveSlopes_def by simp
  ultimately have [M^S] ≤ [f∘N^S] using Real_ZF_1_2_L12
    by simp
  with A1 T show M^R ≤ [f]·(N^R) using Real_ZF_1_1_L4
    by simp
qed
```

A technical lemma aimed used in the proof the candidate for supremum of S is less or equal than any upper bound for S.

```
lemma (in real1) Real_ZF_1_4_L23A:
  assumes A1: f ∈ 𝒮 and A2: N ∈ int  M ∈ int and
  A3: ∀n∈ℤ₊. f(N·n) ≤  M·n
  shows [f]·(N^R) ≤ M^R
proof -
  let M^S = {⟨n, M·n⟩ . n ∈ int}
  let N^S = {⟨n, N·n⟩ . n ∈ int}
  from A1 A2 have T: M^S ∈ 𝒮  N^S ∈ 𝒮  f∘N^S ∈ 𝒮
    using int1.Int_ZF_2_5_L1 int1.Int_ZF_2_1_L11 SlopeOp2_def
    by auto
  moreover from A1 A2 A3 have
    f∘N^S ∼ M^S ∨  M^S + (-(f∘N^S)) ∈ 𝒮₊
    using int1.Int_ZF_2_5_L9 SlopeOp2_def SlopeOp1_def Slopes_def
      BoundedIntMaps_def SlopeEquivalenceRel_def PositiveIntegers_def
      PositiveSlopes_def by simp
  ultimately have [f∘N^S] ≤ [M^S] using Real_ZF_1_2_L12
    by simp
  with A1 T show  [f]·(N^R)≤ M^R using Real_ZF_1_1_L4
    by simp
qed
```

The essential condition to claim that the candidate for supremum of S is greater or equal than all elements of S.

```
lemma (in real1) Real_ZF_1_4_L24:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: x<y  y∈S  and
  A4: N ∈ ℤ₊  M ∈ int and
```

```
  A5: M^R < y·N^R and A6: p ∈ ℤ+
  shows p·M ≤ Γ(S,p·N)
proof -
  from A2 A4 A6 have T1:
    N^R ∈ ℝ+   y∈ℝ   p^R ∈ ℝ+
    p·N ∈ ℤ+   (p·N)^R ∈ ℝ+
    using int_pos_is_real_pos Real_ZF_1_2_L15
    int0.pos_int_closed_mul_unfold by auto
  with A4 A6 have T2:
    p ∈ int   p^R ∈ ℝ   N^R ∈ ℝ   N^R ≠ 0   M^R ∈ ℝ
    using real_int_is_real PositiveSet_def by auto
  from T1 A5 have ⌊(p·N)^R·(M^R·(N^R)^-1)⌋ ≤ ⌊(p·N)^R·y⌋
    using Real_ZF_1_3_L4A Real_ZF_1_3_L7 Real_ZF_1_4_L9
    by simp
  moreover from A1 A2 T1 have ⌊(p·N)^R·y⌋ ≤ Γ(S,p·N)
    using Real_ZF_1_4_L11 by simp
  ultimately have I: ⌊(p·N)^R·(M^R·(N^R)^-1)⌋ ≤ Γ(S,p·N)
    by (rule int_order_transitive)
  from A4 A6 have (p·N)^R·(M^R·(N^R)^-1) = p^R·N^R·(M^R·(N^R)^-1)
    using PositiveSet_def Real_ZF_1_4_L1C by simp
  with A4 T2 have ⌊(p·N)^R·(M^R·(N^R)^-1)⌋ = p·M
    using Real_ZF_1_3_L8 Real_ZF_1_4_L14A by simp
  with I show p·M ≤ Γ(S,p·N) by simp
qed
```

An obvious fact about odd extension of a function $p \mapsto \Gamma(s,p)$ that is used a couple of times in proofs.

```
lemma (in real1) Real_ZF_1_4_L24A:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0 and A2: p ∈ ℤ+
  and A3:
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ+})
  shows h(p) = Γ(S,p)
proof -
  let g = {⟨p,Γ(S,p)⟩. p∈ℤ+}
  from A1 have I: g : ℤ+→int using  Real_ZF_1_4_L12
    by blast
  with A2 A3 show h(p) = Γ(S,p)
    using int0.Int_ZF_1_5_L11 ZF_fun_from_tot_val
    by simp
qed
```

The candidate for the supremum of S is not smaller than any element of S.

```
lemma (in real1) Real_ZF_1_4_L25:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: ¬HasAmaximum(OrderOnReals,S) and
  A3: x∈S and A4:
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ+})
  shows x ≤ [h]
proof -
```

```
  from A1 A2 A3 have
    S ⊆ ℝ  ¬HasAmaximum(OrderOnReals,S)  x∈S
    using Real_ZF_1_2_L23 by auto
  then have ∃y∈S. x<y by (rule Real_ZF_1_2_L27)
  then obtain y where I: y∈S and  II: x<y
    by auto
  from II have
    ∃M∈int. ∃N∈ℤ₊.  x·N^R < M^R ∧ M^R < y·N^R
    using Arthan_Lemma14iii by simp
  then obtain M N where III: M ∈ int  N∈ℤ₊ and
    IV: x·N^R < M^R  M^R < y·N^R
    by auto
  from II III IV have V: x ≤ M^R·(N^R)^-1
    using int_pos_is_real_pos Real_ZF_1_2_L15 Real_ZF_1_3_L4
    by auto
  from A3 have VI: S≠0 by auto
  with A1 A4 have T1: h ∈ 𝒮 using Real_ZF_1_4_L22
    by simp
  moreover from III have N ∈ int  M ∈ int
    using PositiveSet_def by auto
  moreover have ∀n∈ℤ₊. M·n ≤ h(N·n)
  proof
    let g = {⟨p,Γ(S,p)⟩. p∈ℤ₊}
    fix n assume A5: n∈ℤ₊
    with III have T2: N·n ∈ ℤ₊
      using int0.pos_int_closed_mul_unfold by simp
    from III A5 have
      N·n = n·N  and n·M = M·n
      using PositiveSet_def int0.Int_ZF_1_1_L5 by auto
    moreover
    from A1 I II III IV A5 have
      IsBoundedAbove(S,OrderOnReals)
      x<y  y∈S
      N ∈ ℤ₊  M ∈ int
      M^R < y·N^R  n ∈ ℤ₊
      by auto
    then have n·M ≤ Γ(S,n·N) by (rule Real_ZF_1_4_L24)
    moreover from A1 A4 VI T2 have h(N·n) = Γ(S,N·n)
      using Real_ZF_1_4_L24A by simp
    ultimately show M·n ≤ h(N·n) by auto
  qed
  ultimately have M^R ≤ [h]·N^R using Real_ZF_1_4_L23
    by simp
  with III T1 have M^R·(N^R)^-1 ≤ [h]
    using int_pos_is_real_pos Real_ZF_1_1_L3 Real_ZF_1_3_L4B
    by simp
  with V show x ≤ [h] by (rule real_ord_transitive)
qed
```

The essential condition to claim that the candidate for supremum of S is less or equal than any upper bound of S.

```
lemma (in real1) Real_ZF_1_4_L26:
  assumes A1: IsBoundedAbove(S,OrderOnReals) and
  A2: x≤y  x∈S  and
  A4: N ∈ ℤ₊  M ∈ int and
  A5: y·N^R < M^R  and A6: p ∈ ℤ₊
  shows ⌊(N·p)^R·x⌋ ≤ M·p
proof -
  from A2 A4 A6 have T:
    p·N ∈ ℤ₊  p ∈ int  N ∈ int
    p^R ∈ ℝ₊ p^R ∈ ℝ  N^R ∈ ℝ  x ∈ ℝ  y ∈ ℝ
    using int0.pos_int_closed_mul_unfold PositiveSet_def
      real_int_is_real Real_ZF_1_2_L15 int_pos_is_real_pos
    by auto
  with A2 have (p·N)^R·x ≤ (p·N)^R·y
    using int_pos_is_real_pos Real_ZF_1_2_L14A
    by simp
  moreover from A4 T have I:
    (p·N)^R = p^R·N^R
    (p·M)^R = p^R·M^R
    using Real_ZF_1_4_L1C by auto
  ultimately have (p·N)^R·x ≤ p^R·N^R·y
    by simp
  moreover
  from A5 T I have p^R·(y·N^R) < (p·M)^R
    using Real_ZF_1_3_L7 by simp
  with T have p^R·N^R·y < (p·M)^R using Real_ZF_1_1_L9
    by simp
  ultimately have (p·N)^R·x < (p·M)^R
    by (rule real_strict_ord_transit)
  then have ⌊(p·N)^R·x⌋ ≤ ⌊(p·M)^R⌋
    using Real_ZF_1_4_L9 by simp
  moreover
  from A4 T have p·M ∈ int using int0.Int_ZF_1_1_L5
    by simp
  then have ⌊(p·M)^R⌋ = p·M using Real_ZF_1_4_L14
    by simp
   moreover from A4 A6 have p·N = N·p and p·M = M·p
    using PositiveSet_def int0.Int_ZF_1_1_L5 by auto
  ultimately show ⌊(N·p)^R·x⌋ ≤ M·p by simp
qed
```

A piece of the proof of the fact that the candidate for the supremum of S is not greater than any upper bound of S, done separately for clarity (of mind).

```
lemma (in real1) Real_ZF_1_4_L27:
  assumes IsBoundedAbove(S,OrderOnReals)  S≠0 and
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ₊})
  and p ∈ ℤ₊
```

```
  shows ∃x∈S. h(p) = ⌊p^R·x⌋
  using prems Real_ZF_1_4_L10 Real_ZF_1_4_L24A by auto
```

The candidate for the supremum of S is not greater than any upper bound of S.

```
lemma (in real1) Real_ZF_1_4_L28:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0
  and A2: ∀x∈S. x≤y and A3:
  h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ+})
  shows [h] ≤ y
proof -
  from A1 obtain a where a∈S by auto
  with A1 A2 A3 have T: y∈ℝ  h ∈ 𝒮  [h] ∈ ℝ
    using Real_ZF_1_2_L15 Real_ZF_1_4_L22 Real_ZF_1_1_L3
    by auto
  { assume ¬([h] ≤ y)
    with T have y < [h] using Real_ZF_1_2_L28
      by blast
    then have ∃M∈int. ∃N∈ℤ+.  y·N^R < M^R ∧ M^R < [h]·N^R
      using Arthan_Lemma14iii by simp
    then obtain M N where I: M∈int  N∈ℤ+ and
      II: y·N^R < M^R  M^R < [h]·N^R
      by auto
    from I have III: N^R ∈ ℝ+ using int_pos_is_real_pos
      by simp
    have ∀p∈ℤ+. h(N·p) ≤  M·p
    proof
      fix p assume A4: p∈ℤ+
      with A1 A3 I have ∃x∈S. h(N·p) = ⌊(N·p)^R·x⌋
        using int0.pos_int_closed_mul_unfold Real_ZF_1_4_L27
        by simp
      with A1 A2 I II A4 show h(N·p) ≤  M·p
        using Real_ZF_1_4_L26 by auto
    qed
    with T I have [h]·N^R ≤ M^R
      using PositiveSet_def Real_ZF_1_4_L23A
      by simp
    with T III have [h] ≤  M^R·(N^R)^-1
      using Real_ZF_1_3_L4C by simp
    moreover from T II III have M^R·(N^R)^-1 < [h]
      using Real_ZF_1_3_L4A by simp
    ultimately have False using Real_ZF_1_2_L29 by blast
  } then show [h] ≤ y by auto
qed
```

Now we can prove that every nonempty subset of reals that is bounded above has a supremum.

```
lemma (in real1) real_order_complete:
  assumes A1: IsBoundedAbove(S,OrderOnReals)  S≠0
```

```
  shows HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals{a})
proof (cases HasAmaximum(OrderOnReals,S))
  assume HasAmaximum(OrderOnReals,S)
  with A1 show HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals{a})
    using Real_ZF_1_2_L10 IsAnOrdGroup_def IsPartOrder_def
      Order_ZF_5_L6 by simp
next assume A2: ¬HasAmaximum(OrderOnReals,S)
  let h = OddExtension(int,IntegerAddition,IntegerOrder,{⟨p,Γ(S,p)⟩. p∈ℤ₊})
  let r = OrderOnReals
  from A1 have antisym(OrderOnReals)  S≠0
    using Real_ZF_1_2_L10 IsAnOrdGroup_def IsPartOrder_def by simp
  moreover from A1 A2 have ∀x∈S. ⟨x,[h]⟩ ∈ r
    using Real_ZF_1_4_L25 by simp
  moreover from A1 have ∀y. (∀x∈S. ⟨x,y⟩ ∈ r) ⟶ ⟨[h],y⟩ ∈ r
    using Real_ZF_1_4_L28 by simp
  ultimately show HasAminimum(OrderOnReals,⋂a∈S. OrderOnReals{a})
    by (rule Order_ZF_5_L5)
qed
```

Finally, we are ready to formulate the main result: that the construction of real numbers from the additive group of integers results in a complete ordered field.

```
theorem eudoxus_reals_are_reals: shows
  IsAmodelOfReals(RealNumbers,RealAddition,RealMultiplication,OrderOnReals)
  using real1.reals_are_ord_field real1.real_order_complete
    IsComplete_def IsAmodelOfReals_def by simp
```

This completes the construction. It was fun.

end

29 Complex_ZF.thy

theory `Complex_ZF` **imports** `OrderedField_ZF`

begin

The goal of this theory is to define complex numbers and prove that the Metamath complex numbers axioms hold.

29.1 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 oparations of addition and multiplication of complex numbers and the order raletion 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. 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 $f : R \times \{0\} \rightarrow R, f\langle x, 0\rangle = x$ 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, 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.

constdefs
```
  ReCxAdd(R,A,a,b) ≡ A⟨fst(a),fst(b)⟩
```

```
ImCxAdd(R,A,a,b) ≡ A⟨snd(a),snd(b)⟩

CplxAdd(R,A) ≡
{⟨p, ⟨ ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p)) ⟩ ⟩.
p∈(R×R)×(R×R)}

ImCxMul(R,A,M,a,b) ≡ A⟨M⟨fst(a),snd(b)⟩, M⟨snd(a),fst(b)⟩ ⟩

ReCxMul(R,A,M,a,b) ≡
A⟨M⟨fst(a),fst(b)⟩,GroupInv(R,A)(M⟨snd(a),snd(b)⟩)⟩

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)}

ComplexReals(R,A) ≡ R×{TheNeutralElement(R,A)}

CplxROrder(R,A,r) ≡
InducedRelation(SliceProjection(ComplexReals(R,A)),r)
```

The next locale defines proof context and notation that will be used for complex numbers.

locale complex0 =
  fixes R **and** A **and** M **and** r
  assumes R_are_reals: IsAmodelOfReals(R,A,M,r)

  fixes complex (ℂ)
  defines complex_def[simp]: ℂ ≡ R×R

  fixes rone ($\mathbf{1}_R$)
  defines rone_def[simp]: $\mathbf{1}_R$ ≡ TheNeutralElement(R,M)

  fixes rzero ($\mathbf{0}_R$)
  defines rzero_def[simp]: $\mathbf{0}_R$ ≡ TheNeutralElement(R,A)

  fixes one (**1**)
  defines one_def[simp]: **1** ≡ ⟨$\mathbf{1}_R$, $\mathbf{0}_R$⟩

  fixes zero (**0**)
  defines zero_def[simp]: **0** ≡ ⟨$\mathbf{0}_R$, $\mathbf{0}_R$⟩

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

  fixes creal (ℝ)
  defines creal_def[simp]: ℝ ≡ {⟨r,$\mathbf{0}_R$⟩. r∈R}

  fixes 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 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 :: i⇒i (- _ 70)
  defines rneg_def[simp]: - a ≡  GroupInv(R,A)(a)

  fixes lessr (infix <ℝ 68)
  defines lessr_def[simp]:
  a <ℝ b ≡ ⟨a,b⟩ ∈ StrictVersion(CplxROrder(R,A,r))

  fixes cpnf (+∞)
  defines cpnf_def[simp]: +∞ ≡ ℂ

  fixes cmnf (−∞)
  defines cmnf_def[simp]: −∞ ≡ {ℂ}

  fixes cxr (ℝ*)
  defines cxr_def[simp]: ℝ* ≡ ℝ ∪ {+∞,−∞}

  fixes cltrrset (<)
  defines cltrrset_def[simp]:
  < ≡  StrictVersion(CplxROrder(R,A,r)) ∪
  {⟨−∞,+∞⟩} ∪ (ℝ×{+∞}) ∪ ({−∞}×ℝ )
```

29.2 Axioms of complex numbers

In this section we will prove that all Metamath's axioms of complex numbers hold in the complex0 context.

The next lemma lists some contexts that are valid in the complex0 context

```
lemma (in complex0) valid_cntxts: shows
  field1(R,A,M,r)
  field0(R,A,M)
  ring1(R,A,M,r)
  group3(R,A,r)
  ring0(R,A,M)
  M {is commutative on} R
  group0(R,A)
proof -
  from R_are_reals have I: IsAnOrdField(R,A,M,r)
    using IsAmodelOfReals_def by simp
```

```
  then show field1(R,A,M,r) using OrdField_ZF_1_L2 by simp
  then show ring1(R,A,M,r) and I: field0(R,A,M)
    using field1.axioms ring1_def field1.OrdField_ZF_1_L1B
    by auto
  then show group3(R,A,r) using ring1.OrdRing_ZF_1_L4
    by simp
  from I have IsAfield(R,A,M) using field0.Field_ZF_1_L1
    by simp
  then have IsAring(R,A,M) and M {is commutative on} R
    using IsAfield_def by auto
  then show ring0(R,A,M) and M {is commutative on} R
    using ring0_def by auto
  then show group0(R,A) using ring0.Ring_ZF_1_L1
    by simp
qed
```

The next lemma shows the definition of real and imaginary part of complex sum and product in a more readable form using notation defined in `complex0` locale.

```
lemma (in complex0) cplx_mul_add_defs: shows
  ReCxAdd(R,A,⟨a,b⟩,⟨c,d⟩) = a + c
  ImCxAdd(R,A,⟨a,b⟩,⟨c,d⟩) = b + d
  ImCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) = a·d + b·c
  ReCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) =  a·c + (-b·d)
proof -
  let z1 = ⟨a,b⟩
  let z2 = ⟨c,d⟩
  have ReCxAdd(R,A,z1,z2) ≡  A⟨fst(z1),fst(z2)⟩
   by (rule ReCxAdd_def)
  moreover have ImCxAdd(R,A,z1,z2) ≡  A⟨snd(z1),snd(z2)⟩
    by (rule ImCxAdd_def)
  moreover have
    ImCxMul(R,A,M,z1,z2) ≡ A⟨M<fst(z1),snd(z2)>,M<snd(z1),fst(z2)>⟩
    by (rule ImCxMul_def)
  moreover have
    ReCxMul(R,A,M,z1,z2) ≡
    A⟨M<fst(z1),fst(z2)>,GroupInv(R,A)(M⟨snd(z1),snd(z2)⟩)⟩
    by (rule ReCxMul_def)
  ultimately show
    ReCxAdd(R,A,z1,z2) =  a + c
    ImCxAdd(R,A,⟨a,b⟩,⟨c,d⟩) = b + d
    ImCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) = a·d + b·c
    ReCxMul(R,A,M,⟨a,b⟩,⟨c,d⟩) =  a·c + (-b·d)
    by auto
qed
```

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

```
lemma (in complex0) cplx_mul_add_types:
```

```
  assumes A1: z₁ ∈ ℂ   z₂ ∈ ℂ
  shows
  ReCxAdd(R,A,z₁,z₂) ∈ R
  ImCxAdd(R,A,z₁,z₂) ∈ R
  ImCxMul(R,A,M,z₁,z₂) ∈ R
  ReCxMul(R,A,M,z₁,z₂) ∈ R
proof -
  let a = fst(z₁)
  let b = snd(z₁)
  let c = fst(z₂)
  let d = snd(z₂)
  from A1 have a ∈ R  b ∈ R  c ∈ R  d ∈ R
    by auto
  then have
    a + c ∈ R
    b + d ∈ R
    a·d + b·c ∈ R
    a·c + (- b·d) ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 by auto
  with A1 show
    ReCxAdd(R,A,z₁,z₂) ∈ R
    ImCxAdd(R,A,z₁,z₂) ∈ R
    ImCxMul(R,A,M,z₁,z₂) ∈ R
    ReCxMul(R,A,M,z₁,z₂) ∈ R
    using cplx_mul_add_defs by auto
qed
```

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

```
lemma (in complex0) axresscn: shows ℝ ⊆ ℂ
  using valid_cntxts group0.group0_2_L2 by auto
```

Complex 1 is not complex 0.

```
lemma (in complex0) ax1ne0: shows 1 ≠ 0
proof -
  have IsAfield(R,A,M) using valid_cntxts field0.Field_ZF_1_L1
    by simp
  then show 1 ≠ 0 using IsAfield_def by auto
qed
```

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

```
lemma (in complex0) axaddopr: shows CplxAdd(R,A): ℂ × ℂ → ℂ
proof -
  have ∀p ∈ ℂ×ℂ. ⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩
∈ ℂ
    using cplx_mul_add_types by simp
  then have
    {⟨p,<ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))> ⟩. p ∈
ℂ×ℂ}: ℂ×ℂ → ℂ
    by (rule ZF_fun_from_total)
```

```
  then show CplxAdd(R,A): ℂ × ℂ → ℂ using CplxAdd_def by simp
qed
```

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

```
lemma (in complex0) axmulopr: shows CplxMul(R,A,M): ℂ × ℂ → ℂ
proof -
  have ∀p ∈ ℂ×ℂ.
    ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩ ∈ ℂ
    using cplx_mul_add_types by simp
  then have
   {⟨p,⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩⟩.
    p ∈ ℂ×ℂ}: ℂ×ℂ → ℂ by (rule ZF_fun_from_total)
  then show CplxMul(R,A,M): ℂ × ℂ → ℂ using CplxMul_def by simp
qed
```

What are the values of omplex addition and multiplication in terms of their real and imaginary parts?

```
lemma (in complex0) cplx_mul_add_vals:
  assumes A1: a∈R  b∈R  c∈R  d∈R
  shows
  ⟨a,b⟩ + ⟨c,d⟩ = ⟨a + c, b + d⟩
  ⟨a,b⟩ · ⟨c,d⟩ = ⟨a·c + (-b·d), a·d + b·c⟩
proof -
  let S = CplxAdd(R,A)
  let P = CplxMul(R,A,M)
  let p = ⟨ ⟨a,b⟩, ⟨c,d⟩ ⟩
  have S : ℂ × ℂ → ℂ   p ∈ ℂ × ℂ using axaddopr by auto
  moreover have
    S = {⟨p, <ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))>⟩.

    p ∈ ℂ × ℂ}
    using CplxAdd_def by simp
  ultimately have S(p) = ⟨ReCxAdd(R,A,fst(p),snd(p)),ImCxAdd(R,A,fst(p),snd(p))⟩
    by (rule ZF_fun_from_tot_val)
  then show ⟨a,b⟩ + ⟨c,d⟩ = ⟨a + c, b + d⟩
    using cplx_mul_add_defs by simp
  have P : ℂ × ℂ → ℂ   p ∈ ℂ × ℂ using axmulopr by auto
  moreover have
    P = {⟨p, ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩
⟩.
    p ∈ ℂ × ℂ}
    using CplxMul_def by simp
  ultimately have
    P(p) = ⟨ReCxMul(R,A,M,fst(p),snd(p)),ImCxMul(R,A,M,fst(p),snd(p))⟩
    by (rule ZF_fun_from_tot_val)
  then show ⟨a,b⟩ · ⟨c,d⟩ = ⟨a·c + (-b·d), a·d + b·c⟩
    using cplx_mul_add_defs by simp
qed
```

Complex multiplication is commutative.

```
lemma (in complex0) axmulcom: assumes A1: a ∈ ℂ  b ∈ ℂ
  shows a·b = b·a
  using prems cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L4
      field0.field_mult_comm by auto
```

A sum of complex numbers is complex.

```
lemma (in complex0) axaddcl: assumes a ∈ ℂ  b ∈ ℂ
  shows a+b ∈ ℂ
  using prems axaddopr apply_funtype by simp
```

A product of complex numbers is complex.

```
lemma (in complex0) axmulcl: assumes a ∈ ℂ  b ∈ ℂ
  shows  a·b ∈ ℂ
  using prems axmulopr apply_funtype by simp
```

Multiplication is distributive with respect to addition.

```
lemma (in complex0) axdistr:
  assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a·(b + c) = a·b + a·c
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let b_r = fst(b)
  let b_i = snd(b)
  let c_r = fst(c)
  let c_i = snd(c)
  from A1 have T:
    a_r ∈ R  a_i ∈ R  b_r ∈ R  b_i ∈ R  c_r ∈ R  c_i ∈ R
    b_r+c_r ∈ R  b_i+c_i ∈ R
    a_r·b_r + (-a_i·b_i) ∈ R
    a_r·c_r + (-a_i·c_i) ∈ R
    a_r·b_i + a_i·b_r ∈ R
    a_r·c_i + a_i·c_r ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 by auto
  with A1 have a·(b + c) =
    ⟨a_r·(b_r+c_r) + (-a_i·(b_i+c_i)),a_r·(b_i+c_i) + a_i·(b_r+c_r)⟩
    using cplx_mul_add_vals by auto
  moreover from T have
    a_r·(b_r+c_r) + (-a_i·(b_i+c_i)) =
    a_r·b_r + (-a_i·b_i) + (a_r·c_r + (-a_i·c_i))
    and
    a_r·(b_i+c_i) + a_i·(b_r+c_r) =
    a_r·b_i + a_i·b_r + (a_r·c_i + a_i·c_r)
    using valid_cntxts ring0.Ring_ZF_2_L6 by auto
  moreover from A1 T have
    ⟨a_r·b_r + (-a_i·b_i) + (a_r·c_r + (-a_i·c_i)),
    a_r·b_i + a_i·b_r + (a_r·c_i + a_i·c_r)⟩ =
```

```
    a·b + a·c
    using cplx_mul_add_vals by auto
  ultimately show a·(b + c) = a·b + a·c
    by simp
qed
```

Complex addition is commutative.

```
lemma (in complex0) axaddcom: assumes a ∈ ℂ  b ∈ ℂ
  shows a+b = b+a
  using prems cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L4
  by auto
```

Complex addition is associative.

```
lemma (in complex0) axaddass: assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a + b + c = a + (b + c)
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let b_r = fst(b)
  let b_i = snd(b)
  let c_r = fst(c)
  let c_i = snd(c)
  from A1 have T:
    a_r ∈ R  a_i ∈ R  b_r ∈ R  b_i ∈ R  c_r ∈ R  c_i ∈ R
    a_r+b_r ∈ R  a_i+b_i ∈ R
    b_r+c_r ∈ R  b_i+c_i ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 by auto
  with A1 have a + b + c = ⟨a_r+b_r+c_r,a_i+b_i+c_i⟩
    using cplx_mul_add_vals by auto
  also from A1 T have ... = a + (b + c)
    using valid_cntxts ring0.Ring_ZF_1_L11 cplx_mul_add_vals
    by auto
  finally show a + b + c = a + (b + c)
    by simp
qed
```

Complex multiplication is associative.

```
lemma (in complex0) axmulass: assumes A1: a ∈ ℂ  b ∈ ℂ  c ∈ ℂ
  shows a · b · c = a · (b · c)
proof -
  let a_r = fst(a)
  let a_i = snd(a)
  let b_r = fst(b)
  let b_i = snd(b)
  let c_r = fst(c)
  let c_i = snd(c)
  from A1 have T:
    a_r ∈ R  a_i ∈ R  b_r ∈ R  b_i ∈ R  c_r ∈ R  c_i ∈ R
    a_r·b_r + (-a_i·b_i) ∈ R
```

```
    aᵣ·bᵢ + aᵢ·bᵣ ∈ R
    bᵣ·cᵣ + (-bᵢ·cᵢ) ∈ R
    bᵣ·cᵢ + bᵢ·cᵣ ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4  by auto
  with A1 have a · b · c =
    ⟨(aᵣ·bᵣ + (-aᵢ·bᵢ))·cᵣ + (-(aᵣ·bᵢ + aᵢ·bᵣ)·cᵢ),
    (aᵣ·bᵣ + (-aᵢ·bᵢ))·cᵢ + (aᵣ·bᵢ + aᵢ·bᵣ)·cᵣ⟩
    using cplx_mul_add_vals by auto
  moreover from A1 T have
    ⟨aᵣ·(bᵣ·cᵣ + (-bᵢ·cᵢ)) + (-aᵢ·(bᵣ·cᵢ + bᵢ·cᵣ)),
    aᵣ·(bᵣ·cᵢ + bᵢ·cᵣ) + aᵢ·(bᵣ·cᵣ + (-bᵢ·cᵢ))⟩ =
    a · (b · c)
    using cplx_mul_add_vals by auto
  moreover from T have
    aᵣ·(bᵣ·cᵣ + (-bᵢ·cᵢ)) + (-aᵢ·(bᵣ·cᵢ + bᵢ·cᵣ)) =
    (aᵣ·bᵣ + (-aᵢ·bᵢ))·cᵣ + (-(aᵣ·bᵢ + aᵢ·bᵣ)·cᵢ)
    and
    aᵣ·(bᵣ·cᵢ + bᵢ·cᵣ) + aᵢ·(bᵣ·cᵣ + (-bᵢ·cᵢ)) =
    (aᵣ·bᵣ + (-aᵢ·bᵢ))·cᵢ + (aᵣ·bᵢ + aᵢ·bᵣ)·cᵣ
    using valid_cntxts ring0.Ring_ZF_2_L6 by auto
  ultimately show a · b · c = a · (b · c)
    by auto
qed
```

Complex 1 is real. This really means that the pair $\langle 1, 0\rangle$ is on the real axis.

```
lemma (in complex0) ax1re: shows 1 ∈ ℝ
  using valid_cntxts ring0.Ring_ZF_1_L2 by simp
```

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

```
lemma (in complex0) axi2m1: shows i·i + 1 = 0
  using valid_cntxts ring0.Ring_ZF_1_L2 ring0.Ring_ZF_1_L3
  cplx_mul_add_vals ring0.Ring_ZF_1_L6 group0.group0_2_L6
  by simp
```

0 is the neutral element of complex addition.

```
lemma (in complex0) ax0id: assumes a ∈ ℂ
  shows a + 0 = a
  using prems cplx_mul_add_vals valid_cntxts
    ring0.Ring_ZF_1_L2 ring0.Ring_ZF_1_L3
  by auto
```

The imaginary unit is a complex number.

```
lemma (in complex0) axicn: shows i ∈ ℂ
  using valid_cntxts ring0.Ring_ZF_1_L2 by auto
```

All complex numbers have additive inverses.

```
lemma (in complex0) axnegex: assumes A1: a ∈ ℂ
  shows ∃x∈ℂ. a + x  = 0
```

```
proof -
  let ar = fst(a)
  let ai = snd(a)
  let x = ⟨-ar, -ai⟩
  from A1 have T:
    ar ∈ R   ai ∈ R   (-ar) ∈ R   (-ar) ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L3 by auto
  then have x ∈ ℂ using valid_cntxts ring0.Ring_ZF_1_L3
    by auto
  moreover from A1 T have a + x = 0
    using cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L3
    by auto
  ultimately show ∃x∈ℂ. a + x  = 0
    by auto
qed
```

A non-zero complex number has a multiplicative inverse.

```
lemma (in complex0) axrecex: assumes A1: a ∈ ℂ and A2: a≠0
  shows ∃x∈ℂ. a·x = 1
proof -
  let ar = fst(a)
  let ai = snd(a)
  let m = ar·ar + ai·ai
  from A1 have T1: ar ∈ R   ai ∈ R by auto
  moreover from A1 A2 have ar ≠ 0R ∨ ai ≠ 0R
    by auto
  ultimately have ∃c∈R. m·c = 1R
    using valid_cntxts field1.OrdField_ZF_1_L10
    by auto
  then obtain c where I: c∈R and II: m·c = 1R
    by auto
  let x = ⟨ar·c, -ai·c⟩
  from T1 I have T2: ar·c ∈ R  (-ai·c) ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L4 ring0.Ring_ZF_1_L3
    by auto
  then have x ∈ ℂ by auto
  moreover from A1 T1 T2 I II have a·x = 1
    using cplx_mul_add_vals valid_cntxts ring0.ring_rearr_3_elemA
    by auto
  ultimately show ∃x∈ℂ. a·x = 1 by auto
qed
```

Complex 1 is a right neutral element for multiplication.

```
lemma (in complex0) ax1id: assumes A1: a ∈ ℂ
  shows a·1 = a
  using prems valid_cntxts ring0.Ring_ZF_1_L2 cplx_mul_add_vals
    ring0.Ring_ZF_1_L3 ring0.Ring_ZF_1_L6 by auto
```

A formula for sum of (complex) real numbers.

```
lemma (in complex0) sum_of_reals: assumes a∈ℝ  b∈ℝ
  shows
  a + b = ⟨fst(a) + fst(b),0_R⟩
  using prems valid_cntxts ring0.Ring_ZF_1_L2 cplx_mul_add_vals
    ring0.Ring_ZF_1_L3 by auto
```

The sum of real numbers is real.

```
lemma (in complex0) axaddrcl: assumes A1: a∈ℝ  b∈ℝ
  shows a + b ∈ ℝ
  using prems sum_of_reals valid_cntxts ring0.Ring_ZF_1_L4
  by auto
```

The formula for the product of (complex) real numbers.

```
lemma (in complex0) prod_of_reals: assumes A1: a∈ℝ  b∈ℝ
  shows a · b = ⟨fst(a)·fst(b),0_R⟩
proof -
  let a_r = fst(a)
  let b_r = fst(b)
  from A1 have T:
    a_r ∈ R b_r ∈ R  0_R ∈ R  a_r·b_r ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L2 ring0.Ring_ZF_1_L4
    by auto
  with A1 show a · b = ⟨a_r·b_r,0_R⟩
    using cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L2
      ring0.Ring_ZF_1_L6 ring0.Ring_ZF_1_L3 by auto
qed
```

The product of (complex) real numbers is real.

```
lemma (in complex0) axmulrcl: assumes a∈ℝ  b∈ℝ
  shows a · b ∈ ℝ
  using prems prod_of_reals valid_cntxts ring0.Ring_ZF_1_L4
  by auto
```

The existence of a real negative of a real number.

```
lemma (in complex0) axrnegex: assumes A1: a∈ℝ
  shows ∃ x ∈ ℝ. a + x = 0
proof -
  let a_r = fst(a)
  let x = ⟨-a_r,0_R⟩
  from A1 have T:
    a_r ∈ R  (-a_r) ∈ R  0_R ∈ R
    using valid_cntxts ring0.Ring_ZF_1_L3 ring0.Ring_ZF_1_L2
    by auto
  then have x∈ ℝ by auto
  moreover from A1 T have a + x = 0
    using cplx_mul_add_vals valid_cntxts ring0.Ring_ZF_1_L3
    by auto
  ultimately show ∃x∈ℝ. a + x = 0 by auto
```

qed

Each nonzero real number has a real inverse

```
lemma (in complex0) axrrecex:
  assumes A1: a ∈ ℝ   a ≠ 0
  shows ∃x∈ℝ. a · x = 1
proof -
  let R₀ = R-{0_R}
  let a_r = fst(a)
  let y = GroupInv(R₀,restrict(M,R₀×R₀))(a_r)
  from A1 have T: ⟨y,0_R⟩ ∈ ℝ using valid_cntxts field0.Field_ZF_1_L5
    by auto
  moreover from A1 T have a · ⟨y,0_R⟩ = 1
    using prod_of_reals valid_cntxts
    field0.Field_ZF_1_L5 field0.Field_ZF_1_L6 by auto
  ultimately show ∃ x ∈ ℝ. a · x = 1 by auto
qed
```

Our ℝ symbol is the real axis on the complex plane.

```
lemma (in complex0) real_means_real_axis: shows ℝ = ComplexReals(R,A)
  using ComplexReals_def by auto
```

The `CplxROrder` thing is a relation on the complex reals.

```
lemma (in complex0) cplx_ord_on_cplx_reals:
  shows CplxROrder(R,A,r) ⊆ ℝ×ℝ
  using ComplexReals_def slice_proj_bij real_means_real_axis
    CplxROrder_def InducedRelation_def by auto
```

The strict version of the complex relation is a relation on complex reals.

```
lemma (in complex0) cplx_strict_ord_on_cplx_reals:
  shows StrictVersion(CplxROrder(R,A,r)) ⊆ ℝ×ℝ
  using cplx_ord_on_cplx_reals strict_ver_rel by simp
```

The `CplxROrder` thing is a relation on the complex reals. Here this is formulated as a statement that in `complex0` context $a < b$ implies that a, b are complex reals

```
lemma (in complex0) strict_cplx_ord_type: assumes a <_ℝ b
  shows a∈ℝ  b∈ℝ
  using prems CplxROrder_def def_of_strict_ver InducedRelation_def
    slice_proj_bij ComplexReals_def real_means_real_axis
  by auto
```

A more readable version of the definition of the strict order relation on the real axis. Recall that in the `complex0` context r denotes the (non-strict) order relation on the underlying model of real numbers.

```
lemma (in complex0) def_of_real_axis_order: shows
  ⟨x,0_R⟩ <_ℝ ⟨y,0_R⟩ ⟷ ⟨x,y⟩ ∈ r ∧ x≠y
```

```
proof
  let f = SliceProjection(ComplexReals(R,A))
  assume A1: ⟨x,0_R⟩ <ℝ ⟨y,0_R⟩
  then have ⟨ f⟨x,0_R⟩, f⟨y,0_R⟩ ⟩ ∈ r ∧ x ≠ y
    using CplxROrder_def def_of_strict_ver def_of_ind_relA
    by simp
  moreover from A1 have ⟨x,0_R⟩ ∈ ℝ  ⟨y,0_R⟩ ∈ ℝ
    using strict_cplx_ord_type by auto
  ultimately show ⟨x,y⟩ ∈ r ∧ x≠y
    using slice_proj_bij ComplexReals_def by simp
next assume A1: ⟨x,y⟩ ∈ r ∧ x≠y
  let f = SliceProjection(ComplexReals(R,A))
  have f : ℝ → R
    using ComplexReals_def slice_proj_bij real_means_real_axis
    by simp
  moreover from A1 have T: ⟨x,0_R⟩ ∈ ℝ   ⟨y,0_R⟩ ∈ ℝ
    using valid_cntxts ring1.OrdRing_ZF_1_L3 by auto
  moreover from A1 T have ⟨ f⟨x,0_R⟩, f⟨y,0_R⟩ ⟩ ∈ r
    using slice_proj_bij ComplexReals_def by simp
  ultimately have ⟨ ⟨x,0_R⟩, ⟨y,0_R⟩ ⟩ ∈ InducedRelation(f,r)
    using def_of_ind_relB by simp
  with A1 show ⟨x,0_R⟩ <ℝ ⟨y,0_R⟩
    using CplxROrder_def def_of_strict_ver
    by simp
qed
```

The (non strict) order on complex reals is antisymmetric, transitive and total.

```
lemma (in complex0) cplx_ord_antsym_trans_tot: shows
  antisym(CplxROrder(R,A,r))
  trans(CplxROrder(R,A,r))
  CplxROrder(R,A,r) {is total on} ℝ
proof -
  let f = SliceProjection(ComplexReals(R,A))
  have f ∈ ord_iso(ℝ,CplxROrder(R,A,r),R,r)
    using ComplexReals_def slice_proj_bij real_means_real_axis
      bij_is_ord_iso CplxROrder_def by simp
  moreover have CplxROrder(R,A,r) ⊆ ℝ×ℝ
    using cplx_ord_on_cplx_reals by simp
  moreover have I:
    antisym(r)   r {is total on} R   trans(r)
    using valid_cntxts ring1.OrdRing_ZF_1_L1 IsAnOrdRing_def
      IsLinOrder_def by auto
  ultimately show
    antisym(CplxROrder(R,A,r))
    trans(CplxROrder(R,A,r))
    CplxROrder(R,A,r) {is total on} ℝ
    using ord_iso_pres_antsym ord_iso_pres_tot ord_iso_pres_trans
    by auto
```

```
qed
```

The trichotomy law for the strict order on the complex reals.

```
lemma (in complex0) cplx_strict_ord_trich:
  assumes a ∈ ℝ  b ∈ ℝ
  shows Exactly_1_of_3_holds(a<ℝb, a=b, b<ℝa)
  using prems cplx_ord_antsym_trans_tot strict_ans_tot_trich
  by simp
```

The strict order on the complex reals is kind of antisymetric.

```
lemma (in complex0) pre_axlttri: assumes A1: a ∈ ℝ   b ∈ ℝ
  shows a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a)
proof -
  from A1 have Exactly_1_of_3_holds(a<ℝb, a=b, b<ℝa)
    by (rule cplx_strict_ord_trich)
  thus a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a)
    by (rule Fol1_L8A)
qed
```

The strict order on complex reals is transitive.

```
lemma (in complex0) cplx_strict_ord_trans:
  shows trans(StrictVersion(CplxROrder(R,A,r)))
  using cplx_ord_antsym_trans_tot strict_of_transB by simp
```

The strict order on complex reals is transitive - the explicit version of `cplx_strict_ord_trans`.

```
lemma (in complex0) pre_axlttrn:
  assumes A1: a <ℝ b   b <ℝ c
  shows a <ℝ c
proof -
  let s = StrictVersion(CplxROrder(R,A,r))
  from A1 have
    trans(s)   ⟨a,b⟩ ∈ s ∧ ⟨b,c⟩ ∈ s
    using cplx_strict_ord_trans by auto
  then have ⟨a,c⟩ ∈ s by (rule Fol1_L3)
  then show a <ℝ c by simp
qed
```

The strict order on complex reals is preserved by translations.

```
lemma (in complex0) pre_axltadd:
  assumes A1: a <ℝ b and A2: c ∈ ℝ
  shows c+a <ℝ c+b
proof -
  from A1 have T: a∈ℝ  b∈ℝ using strict_cplx_ord_type
    by auto
  with A1 A2 show c+a <ℝ c+b
    using def_of_real_axis_order valid_cntxts
      group3.group_strict_ord_transl_inv sum_of_reals
```

```
    by auto
qed
```

The set of positive complex reals is closed with respect to multiplication.

```
lemma (in complex0) pre_axmulgt0: assumes A1: 0 <ℝ a   0 <ℝ b
  shows 0 <ℝ a·b
proof -
  from A1 have T: a∈ℝ  b∈ℝ using strict_cplx_ord_type
    by auto
  with A1 show 0 <ℝ a·b
    using def_of_real_axis_order valid_cntxts field1.pos_mul_closed
      def_of_real_axis_order prod_of_reals
    by auto
qed
```

The order on complex reals is linear and complete.

```
lemma (in complex0) cmplx_reals_ord_lin_compl: shows
  CplxROrder(R,A,r) {is complete}
  IsLinOrder(ℝ,CplxROrder(R,A,r))
proof -
  have SliceProjection(ℝ) ∈ bij(ℝ,R)
    using slice_proj_bij ComplexReals_def real_means_real_axis
    by simp
  moreover have r ⊆ R×R using valid_cntxts ring1.OrdRing_ZF_1_L1
    IsAnOrdRing_def by simp
  moreover from R_are_reals have
    r {is complete} and IsLinOrder(R,r)
    using IsAmodelOfReals_def valid_cntxts ring1.OrdRing_ZF_1_L1
    IsAnOrdRing_def by auto
  ultimately show
    CplxROrder(R,A,r) {is complete}
    IsLinOrder(ℝ,CplxROrder(R,A,r))
    using CplxROrder_def real_means_real_axis ind_rel_pres_compl
      ind_rel_pres_lin by auto
qed
```

The property of the strict order on complex reals that corresponds to completeness.

```
lemma (in complex0) pre_axsup: assumes A1: X ⊆ ℝ   X ≠ 0 and
  A2: ∃x∈ℝ. ∀y∈X. y <ℝ x
  shows
  ∃x∈ℝ. (∀y∈X. ¬(x <ℝ y)) ∧ (∀y∈ℝ. (y <ℝ x ⟶ (∃z∈X. y <ℝ z)))
proof -
  let s = StrictVersion(CplxROrder(R,A,r))
  have
    CplxROrder(R,A,r) ⊆ ℝ×ℝ
    IsLinOrder(ℝ,CplxROrder(R,A,r))
    CplxROrder(R,A,r) {is complete}
    using cplx_ord_on_cplx_reals cmplx_reals_ord_lin_compl
```

```
    by auto
  moreover note A1
  moreover have s = StrictVersion(CplxROrder(R,A,r))
    by simp
  moreover from A2 have ∃u∈ℝ. ∀y∈X. ⟨y,u⟩ ∈ s
    by simp
  ultimately have
    ∃x∈ℝ. ( ∀y∈X. ⟨x,y⟩ ∉ s ) ∧
    (∀y∈ℝ. ⟨y,x⟩ ∈ s ⟶ (∃z∈X. ⟨y,z⟩ ∈ s))
    by (rule strict_of_compl)
  then show (∃x∈ℝ. (∀y∈X. ¬(x <ℝ y)) ∧
    (∀y∈ℝ. (y <ℝ x ⟶ (∃z∈X. y <ℝ z))))
    by simp
qed

end
```

30 MMI_prelude.thy

theory `MMI_prelude` **imports** `equalities`

begin

In this theory file we define the context in which theorems imported from Metamath are proven and prove the logic and set theory Metamath lemmas that the proofs of Metamath theorems about real and complex numbers depend on.

30.1 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 distribution. 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 stataments 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_logis_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.

30.2 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.

```
consts
   IsASet :: i⇒o (_ isASet [90] 90)

defs
  set_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.

```
locale MMIsar0 =
  fixes real (ℝ)
  fixes complex (ℂ)
  fixes one :: i (1)
  fixes zero :: i (0)
  fixes iunit :: i (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 :: i⇒i (-_ 95)
  defines cneg_def: - a ≡ 0 - a
```

fixes cdiv (**infixl** / 70)
defines cdiv_def: $a / b \equiv \bigcup \{ x \in \mathbb{C}.\ b \cdot x = a \}$
fixes cpnf ($+\infty$)
defines cpnf_def: $+\infty \equiv \mathbb{C}$
fixes cmnf ($-\infty$)
defines cmnf_def: $-\infty \equiv \{\mathbb{C}\}$
fixes cxr ($\mathbb{R}^*$)
defines cxr_def: $\mathbb{R}^* \equiv \mathbb{R} \cup \{+\infty,-\infty\}$
fixes lessr (**infix** $<_{\mathbb{R}}$ 68)
defines lessr_def: $a <_{\mathbb{R}} b \equiv \langle a,b\rangle \in <_{\mathbb{R}}$
fixes cltrrset (<)
defines cltrrset_def:
$< \equiv (<_{\mathbb{R}} \cap \mathbb{R}\times\mathbb{R}) \cup \{\langle -\infty,+\infty\rangle\} \cup$
$(\mathbb{R}\times\{+\infty\}) \cup (\{-\infty\}\times\mathbb{R})$
fixes cltrr (**infix** < 68)
defines cltrr_def: $a < b \equiv \langle a,b\rangle \in <$
fixes lsq (**infix** $\leq$ 68)
defines lsq_def: $a \leq b \equiv \neg (b < a)$

assumes MMI_pre_axlttri:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow (A <_{\mathbb{R}} B \longleftrightarrow \neg(A=B \vee B <_{\mathbb{R}} A))$
assumes MMI_pre_axlttrn:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \wedge C \in \mathbb{R} \longrightarrow ((A <_{\mathbb{R}} B \wedge B <_{\mathbb{R}} C) \longrightarrow A <_{\mathbb{R}} C)$
assumes MMI_pre_axltadd:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \wedge C \in \mathbb{R} \longrightarrow (A <_{\mathbb{R}} B \longrightarrow C+A <_{\mathbb{R}} C+B)$
assumes MMI_pre_axmulgt0:
$A \in \mathbb{R} \wedge B \in \mathbb{R} \longrightarrow (\mathbf{0} <_{\mathbb{R}} A \wedge \mathbf{0} <_{\mathbb{R}} B \longrightarrow \mathbf{0} <_{\mathbb{R}} A\cdot B)$
assumes MMI_pre_axsup:
$A \subseteq \mathbb{R} \wedge A \neq 0 \wedge (\exists x\in\mathbb{R}.\ \forall y\in A.\ y <_{\mathbb{R}} x) \longrightarrow$
$(\exists x\in\mathbb{R}.\ (\forall y\in A.\ \neg(x <_{\mathbb{R}} y)) \wedge (\forall y\in\mathbb{R}.\ (y <_{\mathbb{R}} x \longrightarrow (\exists z\in A.\ y <_{\mathbb{R}} z))))$
assumes MMI_axresscn: $\mathbb{R} \subseteq \mathbb{C}$
assumes MMI_ax1ne0: $\mathbf{1} \neq \mathbf{0}$
assumes MMI_axcnex: $\mathbb{C}$ isASet
assumes MMI_axaddopr: $+ : (\mathbb{C} \times \mathbb{C}) \rightarrow \mathbb{C}$
assumes MMI_axmulopr: $\cdot : (\mathbb{C} \times \mathbb{C}) \rightarrow \mathbb{C}$
assumes MMI_axmulcom: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A \cdot B = B \cdot A$
assumes MMI_axaddcl: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A + B \in \mathbb{C}$
assumes MMI_axmulcl: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A \cdot B \in \mathbb{C}$
assumes MMI_axdistr:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A\cdot(B + C) = A\cdot B + A\cdot C$
assumes MMI_axaddcom: $A \in \mathbb{C} \wedge B \in \mathbb{C} \longrightarrow A + B = B + A$
assumes MMI_axaddass:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A + B + C = A + (B + C)$
assumes MMI_axmulass:
$A \in \mathbb{C} \wedge B \in \mathbb{C} \wedge C \in \mathbb{C} \longrightarrow A \cdot B \cdot C = A \cdot (B \cdot C)$
assumes MMI_ax1re: $\mathbf{1} \in \mathbb{R}$
assumes MMI_axi2m1: $i \cdot i + \mathbf{1} = \mathbf{0}$
assumes MMI_ax0id: $A \in \mathbb{C} \longrightarrow A + \mathbf{0} = A$
assumes MMI_axicn: $i \in \mathbb{C}$

```
  assumes MMI_axnegex: A ∈ ℂ ⟶ ( ∃ x ∈ ℂ. ( A + x ) = 0 )
  assumes MMI_axrecex: A ∈ ℂ ∧ A ≠ 0 ⟶ ( ∃ x ∈ ℂ. A · x = 1)
  assumes MMI_ax1id: A ∈ ℂ ⟶ A · 1 = A
  assumes MMI_axaddrcl: A ∈ ℝ ∧ B ∈ ℝ ⟶ A + B ∈ ℝ
  assumes MMI_axmulrcl: A ∈ ℝ ∧ B ∈ ℝ ⟶ A · B ∈ ℝ
  assumes MMI_axrnegex: A ∈ ℝ ⟶ ( ∃ x ∈ ℝ. A + x = 0 )
  assumes MMI_axrrecex: A ∈ ℝ ∧ A ≠ 0 ⟶ ( ∃ x ∈ ℝ. A · x = 1 )


constdefs
  StrictOrder (infix Orders 65)
  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)


end
```

31 Metamath_interface.thy

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.

The next lemma states that we can use the theorems proven in the MMIsar0 context in the complex0 context. Unfortunately we have to use low level Isabelle methods "rule" and "unfold" in the proof, simp and blast fail on the order axioms.

```
lemma (in complex0) MMIsar_valid:
  shows MMIsar0(ℝ,ℂ,1,0,i,CplxAdd(R,A),CplxMul(R,A,M),
  StrictVersion(CplxROrder(R,A,r)))
proof -
  let real = ℝ
  let complex = ℂ
  let zero = 0
  let one = 1
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset = CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have ℝ ⊆ ℂ using axresscn by simp
  moreover have 1 ≠ 0 using ax1ne0 by simp
  moreover have ℂ isASet by simp
  moreover have  CplxAdd(R,A) : ℂ × ℂ → ℂ
    using axaddopr by simp
  moreover have CplxMul(R,A,M) : ℂ × ℂ → ℂ
    using axmulopr by simp
  moreover have
    ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a· b = b · a
    using axmulcom by simp
  moreover have ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a + b ∈ ℂ
    using axaddcl by simp
  moreover have ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶ a · b ∈ ℂ
    using axmulcl by simp
  moreover have
    ∀a b C. a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
    a · (b + C) = a · b + a · C
    using axdistr by simp
  moreover have ∀a b. a ∈ ℂ ∧ b ∈ ℂ ⟶
        a + b = b + a
    using axaddcom by simp
  moreover have ∀a b C. a ∈ ℂ ∧ b ∈ ℂ ∧ C ∈ ℂ ⟶
      a + b + C = a + (b + C)
    using axaddass by simp
```

```
moreover have
  ∀a b c. a ∈ ℂ ∧ b ∈ ℂ ∧ c ∈ ℂ ⟶ a·b·c = a·(b·c)
  using axmulass by simp
moreover have 1 ∈ ℝ using ax1re by simp
moreover have i·i + 1 = 0
  using axi2m1 by simp
moreover have ∀a. a ∈ ℂ ⟶ a + 0 = a
  using ax0id by simp
moreover have i ∈ ℂ using axicn by simp
moreover have ∀a. a ∈ ℂ ⟶ (∃x∈ℂ. a + x = 0)
  using axnegex by simp
moreover have ∀a. a ∈ ℂ ∧ a ≠ 0 ⟶ (∃x∈ℂ. a · x = 1)
  using axrecex by simp
moreover have ∀a. a ∈ ℂ ⟶ a·1 = a
  using ax1id by simp
moreover have ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ a + b ∈ ℝ
  using axaddrcl by simp
moreover have ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ a · b ∈ ℝ
  using axmulrcl by simp
moreover have ∀a. a ∈ ℝ ⟶ (∃x∈ℝ. a + x = 0)
  using axrnegex by simp
moreover have ∀a. a ∈ ℝ ∧ a≠0 ⟶ (∃x∈ℝ. a · x = 1)
  using axrrecex by simp
ultimately have real ⊆ complex ∧
  one ≠ zero ∧
  complex isASet ∧
  caddset ∈ complex × complex → complex ∧
  cmulset ∈ complex × complex → complex ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶
  cmulset ⟨A, B⟩ = cmulset ⟨B, A⟩) ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶ caddset ⟨A, B⟩ ∈ complex) ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶ cmulset ⟨A, B⟩ ∈ complex) ∧
  (∀A B C.
  A ∈ complex ∧ B ∈ complex ∧ C ∈ complex ⟶
  cmulset ⟨A, caddset ⟨B, C⟩⟩ =
  caddset ⟨cmulset ⟨A, B⟩, cmulset ⟨A, C⟩⟩) ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶
  caddset ⟨A, B⟩ = caddset ⟨B, A⟩) ∧
  (∀A B C.
  A ∈ complex ∧ B ∈ complex ∧ C ∈ complex ⟶
  caddset ⟨caddset ⟨A, B⟩, C⟩ =
  caddset ⟨A, caddset ⟨B, C⟩⟩) ∧
  (∀A B C.
  A ∈ complex ∧ B ∈ complex ∧ C ∈ complex ⟶
  cmulset ⟨cmulset ⟨A, B⟩, C⟩ =
  cmulset ⟨A, cmulset ⟨B, C⟩⟩) ∧
  one ∈ real ∧
  caddset ⟨cmulset ⟨iunit, iunit⟩, one⟩ = zero ∧
  (∀A. A ∈ complex ⟶ caddset ⟨A, zero⟩ = A) ∧
```

```
  iunit ∈ complex ∧
  (∀A. A ∈ complex ⟶ (∃x∈complex. caddset ⟨A, x⟩ = zero)) ∧
  (∀A. A ∈ complex ∧ A ≠ zero ⟶
  (∃x∈complex. cmulset ⟨A, x⟩ = one)) ∧
  (∀A. A ∈ complex ⟶ cmulset ⟨A, one⟩ = A) ∧
  (∀A B. A ∈ real ∧ B ∈ real ⟶ caddset ⟨A, B⟩ ∈ real) ∧
  (∀A B. A ∈ real ∧ B ∈ real ⟶ cmulset ⟨A, B⟩ ∈ real) ∧
  (∀A. A ∈ real ⟶ (∃x∈real. caddset ⟨A, x⟩ = zero)) ∧
  (∀A. A ∈ real ∧ A ≠ zero ⟶ (∃x∈real. cmulset ⟨A, x⟩ = one))
  by simp
moreover have (∀a b. a ∈ real ∧ b ∈ real ⟶
  ⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel))
proof -
  have I:
    ∀a b. a ∈ ℝ ∧ b ∈ ℝ ⟶ (a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a))
    using pre_axlttri by blast
  { fix a b assume a ∈ real ∧ b ∈ real
    with I have (a <ℝ b ⟷ ¬(a=b ∨ b <ℝ a))
      by blast
    hence
      ⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)
      by simp
  } thus (∀a b. a ∈ real ∧ b ∈ real ⟶
      (⟨a, b⟩ ∈ lessrrel ⟷ ¬ (a = b ∨ ⟨b, a⟩ ∈ lessrrel)))
    by blast
qed
moreover have (∀a b c.
  a ∈ real ∧ b ∈ real ∧ c ∈ real ⟶
  ⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel)
proof -
  have II: ∀a b c.  a ∈ ℝ ∧ b ∈ ℝ ∧ c ∈ ℝ ⟶
    ((a <ℝ b ∧ b <ℝ c) ⟶ a <ℝ c)
    using pre_axlttrn by blast
  { fix a b c assume a ∈ real ∧ b ∈ real ∧ c ∈ real
    with II have (a <ℝ b ∧ b <ℝ c) ⟶ a <ℝ c
      by blast
    hence
      ⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel
      by simp
  } thus  (∀a b c.
      a ∈ real ∧ b ∈ real ∧ c ∈ real ⟶
      ⟨a, b⟩ ∈ lessrrel ∧ ⟨b, c⟩ ∈ lessrrel ⟶ ⟨a, c⟩ ∈ lessrrel)
    by blast
qed
moreover have (∀A B C.
  A ∈ real ∧ B ∈ real ∧ C ∈ real ⟶
  ⟨A, B⟩ ∈ lessrrel ⟶
  ⟨caddset ⟨C, A⟩, caddset ⟨C, B⟩⟩ ∈ lessrrel)
  using pre_axltadd by simp
```

```
moreover have (∀A B. A ∈ real ∧ B ∈ real ⟶
  ⟨zero, A⟩ ∈ lessrrel ∧ ⟨zero, B⟩ ∈ lessrrel ⟶
  ⟨zero, cmulset ⟨A, B⟩⟩ ∈ lessrrel)
  using pre_axmulgt0 by simp
moreover have
  (∀A. A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨y, x⟩ ∈ lessrrel) ⟶
  (∃x∈real.
  (∀y∈A. ⟨x, y⟩ ∉ lessrrel) ∧
  (∀y∈real. ⟨y, x⟩ ∈ lessrrel ⟶ (∃z∈A. ⟨y, z⟩ ∈ lessrrel))))
  using pre_axsup by simp
ultimately have
  (∀A B. A ∈ real ∧ B ∈ real ⟶
  ⟨A, B⟩ ∈ lessrrel ⟷ ¬ (A = B ∨ ⟨B, A⟩ ∈ lessrrel)) ∧
  (∀A B C.
  A ∈ real ∧ B ∈ real ∧ C ∈ real ⟶
  ⟨A, B⟩ ∈ lessrrel ∧ ⟨B, C⟩ ∈ lessrrel ⟶ ⟨A, C⟩ ∈ lessrrel) ∧
  (∀A B C.
  A ∈ real ∧ B ∈ real ∧ C ∈ real ⟶
  ⟨A, B⟩ ∈ lessrrel ⟶
  ⟨caddset ⟨C, A⟩, caddset ⟨C, B⟩⟩ ∈ lessrrel) ∧
  (∀A B. A ∈ real ∧ B ∈ real ⟶
  ⟨zero, A⟩ ∈ lessrrel ∧ ⟨zero, B⟩ ∈ lessrrel ⟶
  ⟨zero, cmulset ⟨A, B⟩⟩ ∈ lessrrel) ∧
  (∀A. A ⊆ real ∧ A ≠ 0 ∧ (∃x∈real. ∀y∈A. ⟨y, x⟩ ∈ lessrrel) ⟶
  (∃x∈real.
  (∀y∈A. ⟨x, y⟩ ∉ lessrrel) ∧
  (∀y∈real. ⟨y, x⟩ ∈ lessrrel ⟶ (∃z∈A. ⟨y, z⟩ ∈ lessrrel)))) ∧
  real ⊆ complex ∧
  one ≠ zero ∧
  complex isASet ∧
  caddset ∈ complex × complex → complex ∧
  cmulset ∈ complex × complex → complex ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶
  cmulset ⟨A, B⟩ = cmulset ⟨B, A⟩) ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶ caddset ⟨A, B⟩ ∈ complex) ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶ cmulset ⟨A, B⟩ ∈ complex) ∧
  (∀A B C.
  A ∈ complex ∧ B ∈ complex ∧ C ∈ complex ⟶
  cmulset ⟨A, caddset ⟨B, C⟩⟩ =
  caddset ⟨cmulset ⟨A, B⟩, cmulset ⟨A, C⟩⟩) ∧
  (∀A B. A ∈ complex ∧ B ∈ complex ⟶
  caddset ⟨A, B⟩ = caddset ⟨B, A⟩) ∧
  (∀A B C. A ∈ complex ∧ B ∈ complex ∧ C ∈ complex ⟶
  caddset ⟨caddset ⟨A, B⟩, C⟩ =
  caddset ⟨A, caddset ⟨B, C⟩⟩) ∧
  (∀A B C. A ∈ complex ∧ B ∈ complex ∧ C ∈ complex ⟶
  cmulset ⟨cmulset ⟨A, B⟩, C⟩ = cmulset ⟨A, cmulset ⟨B, C⟩⟩) ∧
  one ∈ real ∧
  caddset ⟨cmulset ⟨iunit, iunit⟩, one⟩ = zero ∧
```

```
    (∀A. A ∈ complex ⟶ caddset ⟨A, zero⟩ = A) ∧
    iunit ∈ complex ∧
    (∀A. A ∈ complex ⟶ (∃x∈complex. caddset ⟨A, x⟩ = zero)) ∧
    (∀A. A ∈ complex ∧ A ≠ zero ⟶
    (∃x∈complex. cmulset ⟨A, x⟩ = one)) ∧
    (∀A. A ∈ complex ⟶ cmulset ⟨A, one⟩ = A) ∧
    (∀A B. A ∈ real ∧ B ∈ real ⟶ caddset ⟨A, B⟩ ∈ real) ∧
    (∀A B. A ∈ real ∧ B ∈ real ⟶ cmulset ⟨A, B⟩ ∈ real) ∧
    (∀A. A ∈ real ⟶ (∃x∈real. caddset ⟨A, x⟩ = zero)) ∧
    (∀A. A ∈ real ∧ A ≠ zero ⟶ (∃x∈real. cmulset ⟨A, x⟩ = one))
    by (rule five_more_conj)
  thus MMIsar0(ℝ,ℂ,1,0,i,CplxAdd(R,A),CplxMul(R,A,M),
    StrictVersion(CplxROrder(R,A,r))) by (unfold MMIsar0_def)
qed
```

In `complex0` context the strict version of the order relation on complex reals is a relation on complex reals.

end

32 MMI_examples.thy

```
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 how a translated Metamath proof looks like. The "known_theorems.txt" file included in the IsarMathLib distribution provides a list of all translated facts.

```
lemma (in MMIsar0) MMI_dividt:
   shows ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( A / A ) = 1
proof -
   have S1: ( A ∈ ℂ ∧ A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( A / A ) = ( A · ( 1 / A ) ) by (rule MMI_divrect)
   from S1 have S2: ( ( A ∈ ℂ ∧ A ∈ ℂ ) ∧ A ≠ 0 ) ⟶
 ( A / A ) = ( A · ( 1 / A ) ) by (rule MMI_3expa)
   from S2 have S3: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( A / A ) = ( A · ( 1 / A ) ) by (rule MMI_anabsan)
   have S4: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( A · ( 1 / A ) ) = 1 by (rule MMI_recidt)
   from S3 S4 show ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( A / A ) = 1 by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_div0t:
   shows ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 0 / A ) = 0
proof -
   have S1: 0 ∈ ℂ by (rule MMI_0cn)
   have S2: ( 0 ∈ ℂ ∧ A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( 0 / A ) = ( 0 · ( 1 / A ) ) by (rule MMI_divrect)
   from S1 S2 have S3: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( 0 / A ) = ( 0 · ( 1 / A ) ) by (rule MMI_mp3an1)
   have S4: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 1 / A ) ∈ ℂ by (rule MMI_recclt)
   have S5: ( 1 / A ) ∈ ℂ ⟶ ( 0 · ( 1 / A ) ) = 0
     by (rule MMI_mul02t)
   from S4 S5 have S6: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶
 ( 0 · ( 1 / A ) ) = 0 by (rule MMI_syl)
   from S3 S6 show ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 0 / A ) = 0 by (rule MMI_eqtrd)
qed

lemma (in MMIsar0) MMI_diveq0t:
   shows ( A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ 0 ) ⟶
 ( ( A / C ) = 0 ⟷ A = 0 )
proof -
   have S1: ( C ∈ ℂ ∧ C ≠ 0 ) ⟶ ( 0 / C ) = 0 by (rule MMI_div0t)
   from S1 have S2: ( C ∈ ℂ ∧ C ≠ 0 ) ⟶
 ( ( A / C ) =
 ( 0 / C ) ⟷ ( A / C ) = 0 ) by (rule MMI_eqeq2d)
   from S2 have S3: ( A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ 0 ) ⟶
```

((A / C) =
(**0** / C) ⟷ (A / C) = **0**) **by** (rule MMI_3adant1)
  have S4: **0** ∈ ℂ **by** (rule MMI_0cn)
  have S5: (A ∈ ℂ ∧ **0** ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (**0** / C) ⟷ A = **0**) **by** (rule MMI_div11t)
  from S4 S5 **have** S6: (A ∈ ℂ ∧ (C ∈ ℂ ∧ C ≠ **0**)) ⟶
((A / C) = (**0** / C) ⟷ A = **0**) **by** (rule MMI_mp3an2)
  from S6 **have** S7: (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) = (**0** / C) ⟷ A = **0**) **by** (rule MMI_3impb)
  from S3 S7 **show** (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
((A / C) = **0** ⟷ A = **0**) **by** (rule MMI_bitr3d)
qed

lemma (**in** MMIsar0) MMI_recrec: **assumes** A1: A ∈ ℂ **and**
  A2: A ≠ **0**
  shows (**1** / (**1** / A)) = A
proof -
  from A1 **have** S1: A ∈ ℂ.
  from A2 **have** S2: A ≠ **0**.
  from S1 S2 **have** S3: (**1** / A) ∈ ℂ **by** (rule MMI_reccl)
  have S4: **1** ∈ ℂ **by** (rule MMI_1cn)
  from A1 **have** S5: A ∈ ℂ.
  have S6: **1** ≠ **0** **by** (rule MMI_ax1ne0)
  from A2 **have** S7: A ≠ **0**.
  from S4 S5 S6 S7 **have** S8: (**1** / A) ≠ **0** **by** (rule MMI_divne0)
  from S3 S8 **have** S9: ((**1** / A) · (**1** / (**1** / A))) = **1**
    by (rule MMI_recid)
  from S9 **have** S10: (A · ((**1** / A) · (**1** / (**1** / A)))) =
(A · **1**) **by** (rule MMI_opreq2i)
  from A1 **have** S11: A ∈ ℂ.
  from A2 **have** S12: A ≠ **0**.
  from S11 S12 **have** S13: (A · (**1** / A)) = **1** **by** (rule MMI_recid)
  from S13 **have** S14: ((A · (**1** / A)) · (**1** / (**1** / A))) =
(**1** · (**1** / (**1** / A))) **by** (rule MMI_opreq1i)
  from A1 **have** S15: A ∈ ℂ.
  from S3 **have** S16: (**1** / A) ∈ ℂ .
  from S3 **have** S17: (**1** / A) ∈ ℂ .
  from S8 **have** S18: (**1** / A) ≠ **0** .
  from S17 S18 **have** S19: (**1** / (**1** / A)) ∈ ℂ **by** (rule MMI_reccl)
  from S15 S16 S19 **have** S20:
    ((A · (**1** / A)) · (**1** / (**1** / A))) =
(A · ((**1** / A) · (**1** / (**1** / A)))) **by** (rule MMI_mulass)
  from S19 **have** S21: (**1** / (**1** / A)) ∈ ℂ .
  from S21 **have** S22: (**1** · (**1** / (**1** / A))) =
(**1** / (**1** / A)) **by** (rule MMI_mulid2)
  from S14 S20 S22 **have** S23:
    (A · ((**1** / A) · (**1** / (**1** / A)))) =
(**1** / (**1** / A)) **by** (rule MMI_3eqtr3)
  from A1 **have** S24: A ∈ ℂ.

```
   from S24 have S25: ( A · 1 ) = A by (rule MMI_mulid1)
   from S10 S23 S25 show ( 1 / ( 1 / A ) ) = A by (rule MMI_3eqtr3)
qed

lemma (in MMIsar0) MMI_divid: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ( A / A ) = 1
proof -
   from A1 have S1: A ∈ ℂ.
   from A1 have S2: A ∈ ℂ.
   from A2 have S3: A ≠ 0.
   from S1 S2 S3 have S4: ( A / A ) = ( A · ( 1 / A ) ) by (rule MMI_divrec)
   from A1 have S5: A ∈ ℂ.
   from A2 have S6: A ≠ 0.
   from S5 S6 have S7: ( A · ( 1 / A ) ) = 1 by (rule MMI_recid)
   from S4 S7 show ( A / A ) = 1 by (rule MMI_eqtr)
qed

lemma (in MMIsar0) MMI_div0: assumes A1: A ∈ ℂ and
    A2: A ≠ 0
   shows ( 0 / A ) = 0
proof -
   from A1 have S1: A ∈ ℂ.
   from A2 have S2: A ≠ 0.
   have S3: ( A ∈ ℂ ∧ A ≠ 0 ) ⟶ ( 0 / A ) = 0 by (rule MMI_div0t)
   from S1 S2 S3 show ( 0 / A ) = 0 by (rule MMI_mp2an)
qed

lemma (in MMIsar0) MMI_div1: assumes A1: A ∈ ℂ
   shows ( A / 1 ) = A
proof -
   from A1 have S1: A ∈ ℂ.
   from S1 have S2: ( 1 · A ) = A by (rule MMI_mulid2)
   from A1 have S3: A ∈ ℂ.
   have S4: 1 ∈ ℂ by (rule MMI_1cn)
   from A1 have S5: A ∈ ℂ.
   have S6: 1 ≠ 0 by (rule MMI_ax1ne0)
   from S3 S4 S5 S6 have S7: ( A / 1 ) = A ⟷ ( 1 · A ) = A
     by (rule MMI_divmul)
   from S2 S7 show ( A / 1 ) = A by (rule MMI_mpbir)
qed

lemma (in MMIsar0) MMI_div1t:
   shows A ∈ ℂ ⟶ ( A / 1 ) = A
proof -
   have S1: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( A / 1 ) =
 ( if ( A ∈ ℂ , A , 1 ) / 1 ) by (rule MMI_opreq1)
```

```
   have S2: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 A = if ( A ∈ ℂ , A , 1 ) by (rule MMI_id)
   from S1 S2 have S3: A =
 if ( A ∈ ℂ , A , 1 ) ⟶
 ( ( A / 1 ) =
 A ⟷
 ( if ( A ∈ ℂ , A , 1 ) / 1 ) =
 if ( A ∈ ℂ , A , 1 ) ) by (rule MMI_eqeq12d)
   have S4: 1 ∈ ℂ by (rule MMI_1cn)
   from S4 have S5: if ( A ∈ ℂ , A , 1 ) ∈ ℂ by (rule MMI_elimel)
   from S5 have S6: ( if ( A ∈ ℂ , A , 1 ) / 1 ) =
 if ( A ∈ ℂ , A , 1 ) by (rule MMI_div1)
   from S3 S6 show A ∈ ℂ ⟶ ( A / 1 ) = A by (rule MMI_dedth)
qed

lemma (in MMIsar0) MMI_divnegt:
   shows ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( - ( A / B ) ) = ( ( - A ) / B )
proof -
   have S1: ( A ∈ ℂ ∧ ( 1 / B ) ∈ ℂ ) ⟶
 ( ( - A ) · ( 1 / B ) ) =
 ( - ( A · ( 1 / B ) ) ) by (rule MMI_mulneg1t)
   have S2: ( B ∈ ℂ ∧ B ≠ 0 ) ⟶ ( 1 / B ) ∈ ℂ by (rule MMI_recclt)
   from S1 S2 have S3: ( A ∈ ℂ ∧ ( B ∈ ℂ ∧ B ≠ 0 ) ) ⟶
 ( ( - A ) · ( 1 / B ) ) =
 ( - ( A · ( 1 / B ) ) ) by (rule MMI_sylan2)
   from S3 have S4: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( ( - A ) · ( 1 / B ) ) =
 ( - ( A · ( 1 / B ) ) ) by (rule MMI_3impb)
   have S5: ( ( - A ) ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( ( - A ) / B ) =
 ( ( - A ) · ( 1 / B ) ) by (rule MMI_divrect)
   have S6: A ∈ ℂ ⟶ ( - A ) ∈ ℂ by (rule MMI_negclt)
   from S5 S6 have S7: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( ( - A ) / B ) =
 ( ( - A ) · ( 1 / B ) ) by (rule MMI_syl3an1)
   have S8: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( A / B ) = ( A · ( 1 / B ) ) by (rule MMI_divrect)
   from S8 have S9: ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( - ( A / B ) ) =
 ( - ( A · ( 1 / B ) ) ) by (rule MMI_negeqd)
   from S4 S7 S9 show ( A ∈ ℂ ∧ B ∈ ℂ ∧ B ≠ 0 ) ⟶
 ( - ( A / B ) ) = ( ( - A ) / B ) by (rule MMI_3eqtr4rd)
qed

lemma (in MMIsar0) MMI_divsubdirt:
   shows ( ( A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ ) ∧ C ≠ 0 ) ⟶
 ( ( A - B ) / C ) =
```

((A / C) - (B / C))
proof -
  have S1: ((A ∈ ℂ ∧ (- B) ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + (- B)) / C) =
((A / C) + ((- B) / C)) **by** (rule MMI_divdirt)
  have S2: B ∈ ℂ ⟶ (- B) ∈ ℂ **by** (rule MMI_negclt)
  from S1 S2 **have** S3: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

((A + (- B)) / C) =
((A / C) + ((- B) / C)) **by** (rule MMI_syl3anl2)
  have S4: (A ∈ ℂ ∧ B ∈ ℂ) ⟶
(A + (- B)) = (A - B) **by** (rule MMI_negsubt)
  from S4 **have** S5: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
(A + (- B)) = (A - B) **by** (rule MMI_3adant3)
  from S5 **have** S6: (A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ⟶
((A + (- B)) / C) =
((A - B) / C) **by** (rule MMI_opreq1d)
  from S6 **have** S7: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A + (- B)) / C) =
((A - B) / C) **by** (rule MMI_adantr)
  have S8: (B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(- (B / C)) = ((- B) / C) **by** (rule MMI_divnegt)
  from S8 **have** S9: ((B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(- (B / C)) = ((- B) / C) **by** (rule MMI_3expa)
  from S9 **have** S10: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(- (B / C)) = ((- B) / C) **by** (rule MMI_3adantl1)
  from S10 **have** S11: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
((A / C) + (- (B / C))) =
((A / C) + ((- B) / C)) **by** (rule MMI_opreq2d)
  have S12: ((A / C) ∈ ℂ ∧ (B / C) ∈ ℂ) ⟶
((A / C) + (- (B / C))) =
((A / C) - (B / C)) **by** (rule MMI_negsubt)
  have S13: (A ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(A / C) ∈ ℂ **by** (rule MMI_divclt)
  from S13 **have** S14: ((A ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A / C) ∈ ℂ **by** (rule MMI_3expa)
  from S14 **have** S15: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(A / C) ∈ ℂ **by** (rule MMI_3adantl2)
  have S16: (B ∈ ℂ ∧ C ∈ ℂ ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ **by** (rule MMI_divclt)
  from S16 **have** S17: ((B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ **by** (rule MMI_3expa)
  from S17 **have** S18: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶
(B / C) ∈ ℂ **by** (rule MMI_3adantl1)
  from S12 S15 S18 **have** S19: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**
) ⟶
((A / C) + (- (B / C))) =
((A / C) - (B / C)) **by** (rule MMI_sylanc)
  from S11 S19 **have** S20: ((A ∈ ℂ ∧ B ∈ ℂ ∧ C ∈ ℂ) ∧ C ≠ **0**) ⟶

((A / C) + ((- B) / C)) =
((A / C) - (B / C)) **by** (rule MMI_eqtr3d)
from S3 S7 S20 **show** ((A $\in \mathbb{C} \wedge$ B $\in \mathbb{C} \wedge$ C $\in \mathbb{C}$) $\wedge$ C $\neq$ **0**) $\longrightarrow$

((A - B) / C) =
((A / C) - (B / C)) **by** (rule MMI_3eqtr3d)
qed

end

33 Metamath_sampler.thy

theory Metamath_sampler **imports** Metamath_interface MMI_Complex_ZF_1

begin

This theory file contains some examples of theorems translated fro Metamath and formulated in the complex0 context.

Metamath uses the set of real numbers extended with $+\infty$ and $-\infty$. The $+\infty$ and $-\infty$ symbols are defined quite arbitrarily as $\mathbb{C}$ and $\{\mathbb{C}\}$, respectively. The next lemma that corresponds to Metamath's renfdisj states that $+\infty$ and $-\infty$ are not elements of $\mathbb{R}$.

```
lemma (in complex0) renfdisj: shows ℝ ∩ {+∞,−∞} = 0
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset= CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
  then have real ∩ {complex, {complex}} = 0
    by (rule MMIsar0.MMI_renfdisj)
  thus ℝ ∩ {+∞,−∞} = 0 by simp
qed
```

The order relation used most often in Metamath is defined on the set of complex reals extended with $+\infty$ and $-\infty$. The next lemma allows to use Metamath's xrltso that states that the < relations is a strict linear order on the extended set.

```
lemma (in complex0) xrltso: < Orders ℝ*
proof -
  let real = ℝ
  let complex = ℂ
  let one = 1
  let zero = 0
  let iunit = i
  let caddset = CplxAdd(R,A)
  let cmulset= CplxMul(R,A,M)
  let lessrrel = StrictVersion(CplxROrder(R,A,r))
  have MMIsar0
    (real, complex, one, zero, iunit, caddset, cmulset, lessrrel)
    using MMIsar_valid by simp
```

```
  then have
    (lessrrel ∩ real × real ∪
    {⟨{complex}, complex⟩} ∪ real × {complex} ∪
      {{complex}} × real) Orders (real ∪ {complex, {complex}})
    by (rule MMIsar0.MMI_xrltso)
  moreover have lessrrel ∩ real × real = lessrrel
    using cplx_strict_ord_on_cplx_reals by auto
  ultimately show < Orders ℝ* by simp
qed

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. S. at al. The Efficient Real Numbers. 2003.

[4] N. D. Megill. Metamath. A Computer Language for Pure Mathematics. 2004. http://us.metamath.org/.